AD-A067 168 WHARTON SCHOOL PHILADELPHIA PA DEPT OF DECISION SCIENCES F/G 9/2
CONTROL PROGRAMS AND DATA STRUCTURES FOR A MULTIPLE ALERTER IMP--ETC(U)
DEC 78 C J SERRA N00014-75-C-0462
UNCLASSIFIED 79-03-07 NL

1 OF 2
AD A067168




Wharton

Department of Decision Sciences

LEVEL II

12

University of
Pennsylvania
Philadelphia PA 19104

DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY PRACTICABLE. THE COPY FURNISHED TO DDC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

(12)

AD A067168

DDC FILE COPY

CONTROL PROGRAMS AND DATA STRUCTURES FOR A MULTIPLE ALERTER IMPLEMENTATION IN A RELATIONAL DATA BASE

Carlos J. Serra

79-03-07

Department of Decision Sciences
The Wharton School
University of Pennsylvania
Philadelphia, PA 19104

Research supported in part by the Office of Naval Research under Contract N00014-75-C-0462.

04 04 002

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER (14) 79-03-07

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER (9)

4. TITLE (and Subtitle) (6)
Control Programs and Data Structures for a Multiple Alerter Implementation in a Relational Data Base.

5. TYPE OF REPORT & PERIOD COVERED
Technical Report, Apr 78 - Mar 79,

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s) (10)
Carlos J. Serra

8. CONTRACT OR GRANT NUMBER(s) (15)
N00014-75-C-0462

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Department of Decision Sciences
The Wharton School
University of Pennsylvania Phila 19104

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
Task NR049-272

11. CONTROLLING OFFICE NAME AND ADDRESS
Office of Naval Research

12. REPORT DATE (11)
Dec 78

13. NUMBER OF PAGES
104

14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) (12) 112p.

15. SECURITY CLASS. (of this report)
Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)
Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)
Distribution unlimited

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)
control programs, data structures, relational data base, multiple alerter implementation, data base management systems

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)
This paper describes the control programs and data structures used to implement multiple alerters in a relational data model environment. Alerters provide data base management systems with the capabilities of dynamically monitoring for the presence of user defined states of the data, and executing some pre-determined action as a consequence of their detection. → next page
(over)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601 |

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

408 757

20. (cont'd)

One of the main considerations in the implementation of an alerter system on a relational data base is the efficiency with which a change to a virtual relation can be detected. Programs are described for the definition, maintenance and evaluation of virtual relations. This work includes details of the use of construction diagrams as a means for their representation, as well as descriptions of the techniques for avoiding unnecessary recomputation and, when required, efficiently performing partial evaluation.

UNIVERSITY OF PENNSYLVANIA

THE MOORE SCHOOL

CONTROL PROGRAMS AND DATA STRUCTURES FOR A MULTIPLE ARRAYS IMPLEMENTATION IN A RELATIONAL DATA BASE

Carlos J. Serra

Presented to the Faculty of the College of Engineering and Applied Science (Department of Computer and Information Science) in partial fulfillment of the requirements for the degree of Master of Science in Engineering.

Philadelphia, Pennsylvania
December, 1976

Peter Buneman

Thesis Supervisor

A. K. Joshi

Graduate Group Chairman

ABSTRACT

CONTROL PROGRAMS AND DATA STRUCTURES FOR A MULTIPLE ALERTER IMPLEMENTATION IN A RELATIONAL DATA BASE

Author Carlos J. Serra

Faculty Supervisor. J. Peter Buneman

This thesis describes the control programs and data structures used to implement multiple alerters in a relational data model environment. Alerters provide data base management systems with the capabilities of dynamically monitoring for the presence of user defined states of the data, and executing some predetermined action as a consequence of their detection.

Complex predicates, describing the states to be monitored, can be viewed as virtual relations; relations whose definition and contents do not exist explicitly in the data base but which are rather expressed in terms of the existing ones. A data sublanguage based on relational algebra seems to be ideally suited for the definition and maintenance of the alerter predicates, for this purpose an extended relational algebra is used and described in this work.

One of the main considerations in the implementation of an alerter system on a relational data base is the efficiency with which a change to a virtual relation can be detected. Programs are described for the definition, maintenance and evaluation of virtual relations. This work includes details of the use of construction diagrams as a means for their representation, as well as descriptions of the techniques for avoiding unnecessary recomputation and , when required, efficiently performing partial evaluation.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 General Description Of The Problem

This thesis describes the implementation of a methodology for the definition of alerters using relational algebra expressions, within the framework of the relational model of data. An alerter is a program which monitors a data base and takes a predefined action when it detects the occurrence of any member of an specified set of conditions. The specification of this set may require the use of non-trivial means of expression, as it may involve several data base entities, or several data operations, or both. Therefore the high flexibility of a relationally complete language, in a relational data base environment, seems to be ideal for the implementation of these alerter definitions.

As the conditions associated to the alerters are expressed in terms of the information contained in the data base, it can intuitively be said that it should be necessary to evaluate the contents of the data base, according to the definitions of the alerters, in order to determine if one of the conditions has been satisfied; satisfaction of these conditions can only be achieved by updating the contents of the data base, since, as we shall see, it is in the spirit of the definition of alerters to take action only when conditions change. On the other hand certain data base updates can be immediately identified, without reference to the contents of the data base, as ones that can not affect the existing alerters; therefore as it is the case that the majority of data base updates, by their nature, should be of no relevance to a set of alerters, it can be seen that an efficient way of avoiding the evaluation of the data base contents in unnecessary occasions is of major importance in implementing an alerter system.

As an illustration of the ideas above, imagine that there exists a data base containing this two relations:

EMPLOYEE: (EMP#,NAME,SALARY,DEPT#)

MANAGERS: (EMP#,DEPT#)

also imagine that it was required to report the name and employee number of new managers as soon as it was reflected in the data base. A relation containing EMP#,NAME,and SALARY for the managers could be obtained as:

```
project EMP#, NAME, EMP# from
        join of EMPLOYEE and MANAGERS on DEPT#
```

It can be seen that the mentioned requirements can be satisfied if this relation is monitored to report any addition that is made to it.

The definition of the alerter, containing the operations and relations needed to express the relation to be monitored can be represented internally as a data structure, which we will later detail and could hypothetically be defined as, say:

```
define alerter ALl =
       monitor for additions,
          indicated by EMP#,
          then report;
       target =
          project((EMP#,NAME,EMP#),
             join((DEPT#),(EMPLOYEE,MANAGERS)));
end;
```

The previous piece of meta-code can be understood as defining an alerter which will monitor and report any addition made to the relation defined as "target", using EMP# as the key to recognize the additions, (they are "indicated by" the appearance of new EMP#'s in the target).

The data structure generated by an equivalent definition would be used to maintain and evaluate the conditions associated to the alerters. This structure should be able to direct the evaluation of the data base, and able to help in the decision of which data base updates are relevant to the existing alerters.

The present thesis is based on a paper by Buneman and Clemons, "Efficiently Monitoring Relational Data Bases", where the aforementioned ideas are presented [BUNE 76], and deals with

the implementation details, algorithms and data structures, of evaluation of alerters in a relational environment. Before discussing this implementation in more detail, we now give some informal description of the nature and use of general alerters.

1.2 Alerters

When working in a data base environment, it would be desirable to ensure the recognition of certain states of the information whenever they are reached due to data base updates. The previous point becomes of much greater importance when the work is taking place in a shared data base environment, and the states to be monitored for can be generated by several users, at different points in time. The following are examples of possible specifications:

1. "Report any checking account whose balance is under $200."
2. "Produce a list of pending deliveries for a client who changes his address."
3. "Issue a warning if a salary is modified to be lower than before."
4. "Report any department where the average salary is over $15000."

Notice that all examples above define an state of the information in the data base and an action to be taken when it is reached.

The programs that monitor the data base for the presence of these states are called alerters. Alerters are expected to exist in a real time environment and perform its functions also in real time; attention should then be paid to every single data base update, thus imposing a great concern on efficiency.

In order to determine if a data base update results in one of the specified states, conventional systems, by and large, require the coding of additional routines specifically for this purpose, usually written in an application language. These routines are usually built by application programmers with no special methodology to follow in the design task. As an expected result they are expensive in both being constructed and run, and very difficult to maintain, they consume too much manpower and do not yield results of the quality that could be expected.

As an step toward the improvement of this situation it has been proposed that a set of alerter rules, provided by the data base administrator, are placed under the responsibility of the data base system for their maintenance and execution. This set of rules can be expressed in terms of a relationally complete language, in a relational data base (RDB) environment. The high flexibility of such kind of language permits the processing of

unanticipated data base updates, as well as proper care of the established set of rules and their application to these updates.

Buneman and Morgan define an alerter as a condition-program pair where the condition may be any predicate involving two consecutive states of the data base [BUNE 75]. The condition describes the state of the data that will be monitored for, while the program part is related to the action to be taken when the condition is satisfied. An alerter should be able to identify certain states of the information when they were reached due to database changes, instead of detecting their presence at any given point in time.

The previous point may be made clearer if we take a look at an example. Let us consider the following command:

"Notify me if any Philadelphia account falls below 500"

The objective of this command is to detect the presence of the state described by "Philadelphia account < 500" in the data base; while also indicating that the action of reporting it to the user should be performed upon its detection.

One obvious alternative that a given user has for the implementation of this command would be to perform this query at a given time and obtain an appropriate answer. On the other hand the same user could define an alerter to perform the function. Under this later alternative the data base management system would have a more active performance being capable of

producing valuable information that has not been specifically requested by a query. In addition, instead of waiting for an user generated test, in many cases with absolutely no regard for the origins of the condition, an alerter would test for the presence of that condition in the expected moment of its generation.

1.2.1 Simple Alerters -

A distinction between simple and complex alerters has been presented in [BUNE 75]. Let us assume that a data base is a collection of records, each of them belonging to one record class, and each record class having a number of component fields. Using this assumption, we have a simple alerter if the condition associated to it is specified by means of expressions involving constants, a single record, and fields of the same record class. This way one or several fields in an arbitrary record of a given record class, as well as additions and deletions of records to a class, can be monitored.

ACCOUNT record class: (ACCT#,NAME,LOCATION,TYPE,AMOUNT)

Figure 1.1: Example of a record class showing record fields

Based on the record class shown in Figure 1.1, it can be seen that the following are simple alerters placed on ACCOUNT. Consider that it was desired to report when:

1. An account falls below 500.

2. The location of an account is changed.

3. An account of type X located at Denver increases its amount by 2000.

4. There is a new account of type S.

5. A New York account is closed.

A common denominator for the previous examples is that all involve expressions and fields of ACCOUNT alone. All of them conform to the specifications given above for simple alerters, as they can be expressed as modifications (1,2,3), additions (4) and deletions (5) to a single record class.

1.2.2 Complex Alerters -

It should be a frequent case that a meaningful alerter has to be expressed in terms of several record classes simultaneously, instead of a single record class; in addition it may also be of interest to define an alerter for a condition expressed in terms of more than one record, or several record

classes, or an aggregate of the information of a given record class. Alerters of greater complexity than the ones defined in the previous section are generated this way, expressing predicates that could not be handled by simple alerting; we will then follow their designation as complex alerters as given in [BUNE 75].

RECORD CLASS:

1. USER: (USER#,USERNAME,CATEGORY,SCHOOL)
2. JOBS: (JOB#,USER#,STATUS,DURATION,LANGUAGE)

Figure 1.2: USER and JOBS record classes

Consider working with data reflecting the use of a computer center. As shown in Figure 1.2, there exists an USER record class which contains information about users of the computer facility, it has the name and number of each user, his category, say student or faculty member, and the school of origin; there is also information about each job, in the JOBS record class, which shows the job and user number, whether the job is finished or not, its duration, and the language used.

Based on the record classes described above, we can now take a look at several examples of some informally defined kinds of complex alerters.

1. "Report any faculty member using APL". In this case we are describing an alerter that involves more than one record class and requires certain knowledge of the data base structure. The condition can become true either because a faculty member starts using APL or because the category of somebody running an APL job is changed to that of faculty member.

2. "Report when the duration of the active jobs average more than T". Even though this alerter requires of only one record class to work it needs to examine a full subset of it, rather than a single record as would be the case of a simple alerter.

3. "Report any user that runs more than ten jobs in a twenty four hours period". The implementation of this alerter would require the maintenance of accessible history for the proper evaluation of the predicate.

All three examples mentioned above represent complex alerters, respectively sensitive to (1) the structure of the data base, (2) data aggregates, and (3) a time or transaction basis. These and other kinds of complex alerters can be expressed, (monitoring for the presence of specific patterns in the data base, for example), while different degrees of complexity can be expected from the standpoint of the evaluation of the data, therefore posing the interesting problem of

efficiently handling general alerters.

The evaluation of the predicates of complex alerters may involve significant amounts of data base processing, since the information pertinent to an alerter may not exist explicitly and therefore had to be evaluated, frequently requiring costly computations. Techniques to efficiently evaluate the alerter predicates, minimize and if possible avoid, data base recomputation are a necessary component of an efficient alerter system. The details of an implementation of these techniques are the subject of this thesis, and they will be discussed more formally ahead.

1.3 Integrity Constraints - A Related Topic

There are several topics that are closely related to the concept of alerters, monitoring data base integrity constraints could perhaps be considered the closer one. A discussion of a general framework for semantic integrity is given in [HAMM 76], while means of efficiently detecting the violation of data base assertions are presented in [HAMM 77].

Using alerter terminology, an integrity constraint could be said to be comparable to an alerter whose programmed action was to reject any data base modification that caused its associated predicate to be true. This is usually approached defining a set of assertions about the integrity of the information in the data

base, these assertions should remain true permanently, and should not be violated by data base updates.

The major distinction between alerters and integrity constraints resides in the fact that we may be interested in defining an alerter to monitor the data base for a perfectly legitimate state whose presence does not imply any error or undesirable situation, while the objective of an integrity constraint is to guard the data base against illegal situations and reject any attempt to generate them.

It could be said that an integrity constraint deals with the potential resulting state of the data base after a modification has been attempted, if this state, based on the defined assertions, is found to be illegal then the modification would be rejected. This is another major distinction between alerters and integrity constraints, which can be considered a consequence of the previous one; in general an alerter is only interested in states that had not been detected before, in other words, an alerter should ignore a condition of the information that was true before the application of the update that is being evaluated. It can be seen that alerters need to keep track of previous detections, ("involving two consecutive states of the data base", quoting from the definition of alerters given before), while a similar integrity constraint has no need of this kind of history as any attempt to violate the assertion should have been prevented.

As an illustration of the previous point, suppose that we wanted to be notified whenever somebody reached the age of 21, it would be preferable that we were only notified of those cases that, because they have just turned 21, were not reported in previous instantiations of the alerter.

There exists, of course, a class of integrity constraints that has to be defined in terms of consecutive states of the information, (salary increases have to be of less than 20%, as an example of a simple case), nevertheless these states are used in a different frame of mind since the assertion needs to be expressed in terms of the transition of the information itself, as this is considered the actual monitorable event.

Some alerter predicates would have difficult expression as integrity assertions, as could be the case for conditions involving a sequence of transactions or a given time period. For example, consider monitoring a bank data base to "report those accounts which maintain a balance under $200 while more than ten checks are cashed", or "report those accounts whose balance was under $100 at closing time". The problem of their implementation as integrity constraints would reside in how to perform the deferred selection of the transaction to be rejected, and while they might lack meaning if expressed as integrity assertions, they certainly can be meaningful alerter predicates.

Alerters do not have a direct effect on the contents of the data base, while this is not the case of integrity constraints, as a result, several problems, in particular consistency problems in a concurrent user environment, are less critical in an alerter implementation. A discussion of an implementation of aspects related to integrity constraints, and general transaction management problems, can be found in [ASTR 76]. In general we would expect alerters to be able to express a more complex class of predicates, and to be at a higher level of implementations than integrity constraints.

CHAPTER 2

RELATIONAL DATA BASES

2.1 The Relational Data Model

In a series of papers Codd introduced the relational model of data in an effort to help in the solution of various data base management problems [CODD 70]. Some of the objectives of this work, as stated by Codd, were:

1. to provide a high degree of data independence;
2. to provide a simple community view of the data;
3. to introduce a theoretical foundation in data base management;
4. to provide the user with a high level, nonprocedural data sublanguage for accessing data.

The relational data model is a formal model for representing relationships among attributes of an entity set and the association between entity sets [CODD 70]. Given sets

S1,S2,..,Sn (not necessarily distinct), R is a relation on these n sets if it is a set of n-tuples each of which has its first element from S1, the second element from S2, and so on. More concisely, R is a subset of the Cartesian product S1*S2*...*Sn.

For convenience, relations are usually represented as a table where each row represents one of its n-tuples (or simply one of its tuples), and each column corresponds to a domain of the relation, and is called an attribute. The number of tuples of a relation is called the cardinality of the relation. The number of attributes of the relation is called the degree of the relation. Relations of degree one, two, and three are called unary, binary, and ternary respectively, relations of degree n are called n-ary relations. Figure 2.1 shows an example of a relation of degree 4.

EMPLOYEE

EMP#	NAME	SALARY	DEPT#
010	JONES	12000	1
020	SMITH	15000	1
040	ADAMS	11500	2
060	WILLIAMS	12500	1
070	SPENCER	12000	2
080	THOMAS	14500	1
090	DAY	11000	2

Figure 2.1: The EMPLOYEE relation

From the user point of view, a relational data base management system organizes and maintains the data according to the relational data model, (This does not imply that the actual

file structure has to resemble that of a table or any similar rectangular organization, it rather implies the presence of the capability to handle the data base as the set of relations as it is conceptualized).

Since relations are sets, the data language provided by such system has to be capable of performing all of the usual set operations, (the result of the application of these operations may not be a relation; for example the union of two relations of different degrees is not a relation). We will restrict ourselves to those operations which result in valid relations. These relational operations allow an user to create new relations based on existing ones. We will refer to a relation existing in the data base as a base relation; every other relation derived from these ones, using the relational operators, will be referred to as a virtual relation; these can either be combinations or subsets of the base relations.

2.2 Relational Algebra

The relational operators can be described using either the relational algebra or the relational calculus. Codd defines a set of high level relational algebra operations and shows that this set is relationally complete [CODD 72]. A language is said to be relationally complete if it possesses the property that any relation definable by means of calculus expressions may be retrieved via suitable statements in that language.

In the relational algebra mentioned above, we have two different kind of operations: traditional set operations and special relational operations, in the first group we have union, intersection, difference and an extended Cartesian product, while in the second group we find projection, join, division and selection.

For the operations union, intersection and difference the relations involved have to be union-compatible; we say that two relations are union-compatible if they are of the same degree, and the ith attribute of the one is obtained from the same domain as the ith attribute of the other.

1. UNION(A,B): The union of two compatible relations A and B is the set of all tuples t that belong to either A or B.

2. INTERSECTION(A,B): The intersection of two compatible relations A and B is the set of all tuples t that belong to both A and B.

3. DIFFERENCE(A,B): The difference between two compatible relations A and B (in that order) is the set of all tuples t belonging to A and not to B.

4. EXTENDED CARTESIAN PRODUCT(A,B): The extended Cartesian product of two relations A and B is the set of all tuples t such that, for all a belonging to A and all b belonging to B, t is the concatenation of a and

b.

5. PROJECT(A,X): A projection of a relation A is obtained selecting some specified attributes X, from A and eliminating the others, the duplicated tuples that might appear are omitted from the resulting relation.

6. JOIN(A,B,X,Y): If two relations A and B have common attributes, X in A and Y in B, they may be joined over those attributes; the result of this operation is a new relation of tuples t created as follows: if two tuples a and b, belonging to A and B respectively, share the selected common attributes, they are joined together into a new tuple t formed by the common attributes, the remaining attributes of a, and the remaining attributes of b.

7. DIVIDE(A,B,Y,Z): Given a binary relation A and an unary relation B, "let the dividend A have attributes X and Y and let the divisor B have attribute Z, and let Y and Z be defined on the same underlying domain. Then the divide operation - DIVIDE A BY B OVER Y AND Z - produces a quotient defined on the same domain as X; a value x will appear in the quotient if and only if the pair <x,y> appears in A for all values y appearing in B" [DATE 75].

8. SELECT(A,X,pr): The select operation, applied to a relation A, generates a subset of A formed by those tuples for which a specified predicate, involving some attributes X in A, is satisfied. In order to avoid any misunderstanding, it should be noted that SELECT is being used in the sense of the restriction operation defined in [CODD 70], since in the data sublanguage SEQUEL the same term defines an operation similar to PROJECT.

CHAPTER 3

ALERTERS IN A RELATIONAL DATA BASE

3.1 Terminology

The problem of implementing alerters in a relational data base environment is discussed by Buneman and Clemons in their paper [BUNE 76], avoiding recomputation, partial evaluation, and related data structures are among the topics analyzed. The development of this Thesis and the terminology that will be described here, were based on the ideas presented in this paper. The general relational terminology that will be used, partially outlined in the previous chapter, is based in the work of E. F. Codd, his original paper [CODD 70], and [CODD 72].

We will know as base relations those which are explicitly defined in the data base schema, every other relation, obtainable from these by means of relational algebra operations will be called a virtual relation. A relation with an associated alerter will be known as a target relation. A target relation can either be base or virtual according to the alerter definition.

Let us suppose that we have the data base schema shown in Figure 3.1, containing information about suppliers, parts, and an indication of part quantities for each supplier. Let us also suppose that the part name should be reported, for any part for which any QTY is greater than 500.

BASE RELATION:

SUPPLIERS: (SUPP#,NAME,ADDRESS,CITY)

PARTS: (PART#,PART-NAME,DESCRIPTION)

SUP-PART: (SUPP#,PART#,QTY)

FIGURE 3.1: Supplier/Part data base schema

In this case SUPPLIER, PARTS, and SUP-PART are all base relations as they explicitly exist in the data base schema. On the other hand, a virtual relation named, say, VR, could be defined in such a way that it would contain the names of those parts for which the mentioned condition, QTY > 500, is true.

VR can be constructed to be a target relation, using relational algebra operations, in the following way: select those tuples of SUP-PART where QTY is greater than 500, join them with PARTS using PART# as the common attribute, and finally project PART-NAME from the result.

A directed graph, describing the generation of a virtual relation from a set of base relations, is called a construction diagram. A construction diagram shows which operations are needed to create the virtual relation, and imposes a partial ordering on their execution. A construction diagram for VR is shown in Figure 3.2.

Alerters will be defined to monitor the contents of the target relations. While monitoring these target relations, an alerter will be looking for possible additions and deletions of tuples. These target relations, if virtual, will be defined by means of construction diagrams, and if necessary, actually constructed to evaluate a given data base update. We will then make a distinction between add-alerters and delete-alerters: an add-alerter will monitor the target relation for addition of tuples while a delete-alerter will do the same for deletions.

For two consecutive states of a relation, before and after being updated, tuples will be considered added or deleted as indicated by an specified set of attributes, in other words, additions or deletions will be identified by comparing the projection of those attributes from the final state against a similar projection of the previous one. We will use the terms ASA and DSA as a short-notation to denote respectively addition and deletion indicated by a set of attributes.

Figure 3.2:

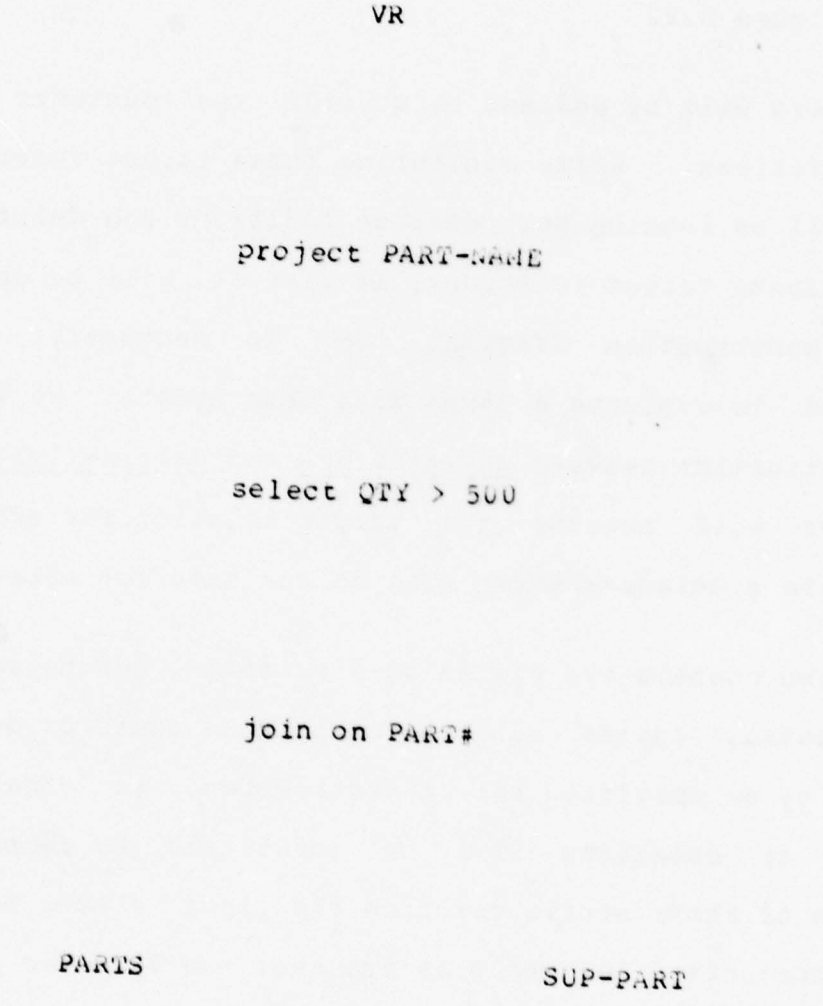

Figure 3.2: A construction diagram for VR

As an illustration of the preceding point, imagine that the address of a given supplier has been modified in our relation of Figure 3.1, strictly speaking the resulting tuple is a new one that did not exist before, yet we can be interested in identifying additions to this relation only when we find a tuple with a SUPP# that was not in the relation before.

When the data base reaches an state whose occurrence is being monitored for by an alerter we say that the alerter has been triggered. As a result of this, some action will take place, ranging from simple reporting to more complex operations on any of the existing relations.

3.2 Relational Algebra - Conceptual Extensions

In a relational data base environment alerters should be expressed in terms of relations and relational algebra operations applied to these relations. Other than the information pertinent to the construction diagram, it should be our goal to express an alerter exclusively in relational algebra terms.

When defining an alerter in these terms, there are a number of virtual relations that can not be constructed using exclusively the operations described in Chapter 2. One very common and frequently required element in an alerter definition is some arithmetical expression involving one tuple, or an

aggregate of tuples in the relation. The algebra defined in Chapter 2 does not possess any kind of arithmetical operations, therefore as we are interested in maximizing our alerter definition capabilities, we have made some additions to the relational operators that we will be using from now on. Even though this additions, by all means, are only a subset of the arithmetical and aggregate operations that could be added to a relational language, we feel that their incorporation to our set of operations will sensibly broaden the application range of our alerters, and will serve our purposes as to characterize the treatment that should be given to similar operations.

We will now define the operations that will be added to our relational algebra in a very similar way to that of Chapter 2.

The action of applying any of the operations in this extended relational algebra will result in another relation, which, in turn, could be operated again as would be required in a construction diagram.

1. MAXIMUM(A,X): The maximum of relation A is a tuple for which a set of specified attributes X has the maximum value.

2. MINIMUM(A,X): The minimum of relation A is a tuple for which a set of specified attributes X has the minimum value.

3. AVERAGE(A,X): The average of relation A for one of its numerical attributes X, is a tuple containing the average and sum of the attribute, and a count, for the tuples in the relation.

4. COUNT(A,X): Given a relation A and a set of attributes X, the count of A is a projection of A on these attributes, each tuple concatenated with the respective count of original tuples in A sharing the projected attributes.

5. BREAK DOWN TOTALS(A,(X,Y)): Given a relation A, a set of attributes X, and a numerical attribute Y, the break down totals of A is a projection of A on the attributes X, where each tuple is concatenated with a count and a sum of Y for the original tuples in A sharing attributes X. As an example consider the schema in Figure 3.1 and imagine that we needed to obtain the sum of QTY for each PART# in SUPP-PART. This could be specified as the break-down-totals of QTY in SUPP-PART by PART#, (BRKDWTOT(SUPP-PART,(QTY,PART#))).

6. CHANGES(A,B,X,Y): Given two relations A and B, union compatible, and a set of common attributes, X in A and Y in B, changes is equivalent to the JOIN operation, restricting the result only to those tuples where there is a difference between any of the correspondent non-common attributes of A and B. We will use this

operation to detect the changes suffered by a relation, since CHANGES will use consecutive states of the relation as its parameters, (before and after updates). In this manner we achieve the capability of expressing alerters to monitor transitions of the data. For example: "report any employee transfered from sales to maintenance".

Now that we have complemented the relational algebra that we will be using, we can define the the set of parameters that are needed in our implementation for the proper execution of each operation; we will make use of the following list for their definition:

	OPERATION	PARAMETERS
1.	UNION :	rel-1 , rel-2;
2.	DIFRNCE:	rel-1 , rel-2;
3.	INTRSCT:	rel-1 , rel-2;
4.	XCPROD :	rel-1 , rel-2;
5.	PROJECT:	rel-1 , att-1;
6.	RCOUNT :	rel-1 , att-1;
7.	AVERAGE:	rel-1 , att-1;
8.	BRDWTOT:	rel-1 , att-1;
9.	MAXTUPL:	rel-1 , att-1;
10.	MINTUPL:	rel-1 , att-1;
11.	JOIN :	rel-1 , rel-2 , att-1 , att-2;

12. DIVIDE : rel-1 , rel-2 , att-1 , att-2;
13. CHANGES: rel-1 , rel-2 , att-1 , att-2;
14. SELECT : rel-1 , att-1 , fn;

For each operation, this list indicates the relations it needs to operate properly, and the sets of attributes, if any, that are required. For the SELECT operation, a conditional function, that operates on a tuple at a time and returns a truth value, has to be defined.

3.3 Some Examples Of Alerter Definitions

In order to clarify ideas and using the terminology that has been described so far, it could be of benefit to present some cases of different alerter definitions.

It is our intention to use these examples as tools to introduce several characteristics of the construction diagrams, that will be part of our implementation. For the purpose of defining the contents of the construction diagram, we will make use of a meta-language to express the operations and necessary related information. All following examples will be based on the data base schema depicted previously in Figure 3.1.

1. "Report the PART-NAME for any part for which QTY > 500 for any supplier". The construction diagram for this example was shown in Figure 3.2, and could be defined as follows:

```
define alerter AL1=
       monitor for additions,
          indicated by PART-NAME,
          then report;
       target=
          project((PART-NAME),
             select((QTY > 500),
                join((PART#),(PART#),
                   PARTS,
                   SUP-PART)));
end;
```

In this case AL1 has been defined as an add-alerter, where the addition of tuples will be indicated by the detection of new PART-NAMEs in the target relation. "Report" is specified as the action to be taken if the alerter is triggered, (we shall see that this is the standard alerter action, which prints the projection of the "indicating" attributes from the added tuples). Any function, expressible by means of display and relational operations, can be defined to be the action associated with the alerter. Finally, the target relation is defined in terms of relational operators and the existing base relations.

2. If we consider the following alerter definition:

```
define alerter AL2=
       monitor for additions,
          indicated by PART-NAME,
          then report;
       target=
          select((QTY > 500),
             join((PART#),(PART#),
                PARTS,
                SUPP-PART));
end;
```

Based on our previous definition of addition and deletion of tuples as indicated by a set of attributes, ASA and DSA, AL2

is equivalent to AL1, with the added advantage of having omitted one step in the construction of the target relation.

3. "Report the supplier NAME and PART-NAME for new parts in the list of any supplier". This alerter could be defined as:

```
define alerter AL3=
       monitor for additions,
          indicated by NAME,PART-NAME,
          then report;
       target=
          join((SUPP#),(SUPP#),
             join((PART#),(PART#),
                PARTS,
                SUP-PART),
             SUPPLIER);
end;
```

If we observe that pieces of AL3 are shared by the previous two alerter definitions, it should be possible to define them as:

```
define node ND1=
       join((PART#),(PART#),
          PARTS,
          SUP-PART);
end;
```

and

```
define alerter AL3=
       monitor for additions,
          indicated by NAME,PART-NAME,
          then report;
       target=
          join((SUPP#),(SUPP#),
             ND1,
             SUPPLIER);
end;
```

AL1 and AL2 should then be modified accordingly. It should be noted that rather than a simple ease of expression, the goal of this node definition is to allow the actual sharing of the nodes in the construction diagram by

more than one alerter. Besides savings in the construction diagram representation, it would be translated into added efficiency since if a given virtual relation had to be constructed for more than one alerter, it would only have to be done once. In the following example we show how AL2 could be modified as an example of an action different than "report".

```
define alerter AL4=
       monitor for additions,
          indicated by PART-NAME,
          then print(join((SUPP#),(SUPP#),
                          target,SUPPLIER));
       target=
          select((QTY > 500),
             ND1);
end;
```

We define this way an alerter similar to AL2, since they only differ in that AL4 executes a different action when the alerter is triggered. In this case the contents of the whole target relation are processed by a join operation, whose result will be the printed data. This is only a trivial example of an action different from reporting associated to the alerter, which is expressed in relational algebra terms.

3.4 Avoiding Recomputation

A data base update can generate major changes to the contents of the relations to which it is applied, nevertheless,

it can have absolutely no effect on a target relation, even if the target is expressed in terms of the updated relations. Furthermore it could be the case that even though the target relation did get modified, the modification was irrelevant to the associated alerter, since the alerter was only defined to monitor either additions or deletions.

Certain data base update can be recognized to be of no relevance to the existing alerters without having to examine the contents the data base, we will refer to them as readily ignorable updates, RIU's, [BUNE 76]. This can be achieved by comparing the characteristics of the update,(relations involved, whether it was an addition or a deletion, and the fields involved if it was a tuple update), with the characteristics of those updates considered to be potentially relevant to the existing alerters; if no coincidence is found, then the update is considered a RIU.

For each base relation and the alerters in which they are involved, the set of potentially relevant updates can be defined by examining the target relations and the operations needed to construct them, and will be specified in terms of the relevance to the alerter of additions, deletions and modification of each attribute. This set can be represented as a vector, henceforth known as relevance vector [BUNE 76].

We will define a relevance vector as: For a relation of degree n, involved in an alerter definition, a vector of length n+2 will be known as its relevance vector if it is such that: (i) the first bit is set to one if addition of tuples can be relevant to the alerter; (ii) the second bit is set to one if deletion of tuples can relevant to the alerter; (iii) for k > 2, the kth bit is set to one if a modification to the (k-2)th attribute can be relevant to the alerter.

As an illustration let us consider the example #3 of the previous section. Alerter AL3 has the following construction diagram:

AL3:(indicated by NAME,PART-NAME)
join on SUPP#

join on PART#

PARTS SUP-PART SUPPLIER

The relevance vectors for the relations PARTS, SUP-PART, and SUPPLIER, reflecting their use in AL3, are: (1 0 1 1 0), (1 0 1 1 0), and (1 0 1 1 0 0), respectively. They indicate that only addition of tuples to the relations and modifications to the attributes SUPP#, PART#, and PART-NAME, can be of relevance to the alerter, while modifications to DESCRIPTION, QTY, ADDRESS, and CITY can not possibly modify the target in a way of interest to AL3.

3.5 Partial Evaluation Using The Construction Diagram

We have said that a data base update can be recognized as a RIU without any reference to the contents of the data base. On the other hand, if an update can not be recognized as such, the construction of the virtual relations needed to evaluate the target has to be started in order to determine how it is affected by the update. Nevertheless, after some of the virtual relations have been constructed and examined, it can be possible to conclude that the update will not affect the target in a relevant way, and, therefore, no further evaluation is necessary. As an example, let us consider that we have defined a target relation as follows:

```
target=
   join((SUPP#),(SUPP#),
      select((QTY > 500),
         SUP-PART),
      SUPPLIER);
```

According to this target definition, the modification of the attribute QTY in a tuple of SUP-PART can not be considered a RIU to any add or delete-alerter associated to the target. In addition, let us also suppose that the alerter was defined as an add-alerter, according to SUPP# and PART#, and that a given QTY is modified to be, say, 400; after the left portion of the construction diagram, corresponding to the select operation, has been evaluated, it is possible to observe that since no tuple has been added to the virtual relation at this node, it will not be possible for one to be added to the target as a result.

In the following Chapter we will discuss the details of our implementation of the functions to detect RIU's and perform partial evaluation.

CHAPTER 4

IMPLEMENTATION DETAILS

4.1 Introduction

The present Chapter describes the algorithms and data structures that were used to implement the ideas we have discussed so far. The Alerter was implemented in the DECSYSTEM-10 computer at the Wharton School of The University of Pennsylvania, using the POP-10 programming language [BURS 71], [DAVI 74]. In this implementation the relations are processed as a list of tuples whose head contains a definition of the relation name, number and characteristics of the attributes, and the functions to handle them. Each tuple is stored as a POP-record structure. The purpose of the programs was not primarily to handle a relational data base, but to explore the techniques described here.

4.2 Computing The Relevance Vectors

When an alerter is defined it is indicated whether it will be an add-alerter or a delete-alerter, and according to which attributes the addition and removal of tuples will be recognized, therefore, a relevance vector reflecting the alerter definition can be assigned to the target relation. In doing so we would achieve the capability to detect additions and deletions to the target, and, if any, whether they are relevant or not; in order to accomplish this detection the contents of the target relation should be available.

One of our main concerns is to minimize the number of occasions in which we need to construct the virtual relations. Using the construction diagram, the sooner we detect that an update will not influence a target relation in a meaningful way, then, as a result, the number of virtual relations that have to be constructed will accordingly be smaller. This minimization can be assisted by the inspection of the relevance of each intermediate step in the construction of a virtual relation, since a relevance vector can be assigned to all the relations in a construction diagram, based on the relevance vector of the target.

We shall now describe how the relevance vector is computed for each node of the construction diagram. First we will define partially the data structure associated to each node.

1. PRE(n): A list that indicates which nodes are located immediately below n in the construction diagram, and contain the relations, operands, that should be handled by OPR(n) defined below. The length of this list can either be one or two depending on whether OPR(n) is an unary or a binary operation; in the case that the node is associated to a base relation PRE(n) will be assumed to be null.

2. OPR(n): A list defined the following way: (i) RATOR: the head of the list, will contain the extended relational algebra operation needed to construct the relation associated to this node; if the node is associated to a base relation a base relation signal will take this place to indicate it; (ii) ATTR: the tail of the list, will contain a list of the attributes that will be used by the operation, for each relation in the nodes in PRE(n); this list will be null for base relations and traditional set operations.

3. RLV(n): The relevance vector associated to this node, will indicate the relevance of the updates applied to this relations.

4. EQV(n): Contains, for each element in PRE(n), a list of the correspondence between the attributes of each of them and the attributes of n; this correspondence indicates from which attributes of the operands are

taken the attributes of the result.

For a given node n, the relevance vector for each node in PRE(n) can be determined based on RLV(n) and OPR(n), (relevance vector, operator, and attributes of the operands). The table of Figure 4.1 shows, for the comparison of two consecutive states of a relation: (i) what kinds of updates to the operands can cause additions of tuples to the result; (ii) what kinds of updates to the operands can cause deletions of tuples from the result; and (iii) what kinds of updates to the operands can cause alteration in the contents of the attributes selected to control addition or deletion of tuples, ASA and DSA.

Using the rules in this table, we can define a function to compute the relevance vector of the nodes in the construction diagram of a target relation, in general, of any virtual relation in a construction diagram. Therefore, we now define the function RLVCOMP to operate on a given node, N, and a relevance vector, RLVL; it will also make use of the function RULES which will apply the rules in the table and will operate on OPR(N), RLV(N), EQV(N) and an indication of whether the rules are being applied to the first or second operand, if there is a second one. RULES will compute a relevance vector for each node in PRE(N), while RLVCOMP will use them to update RLV in all the nodes in the construction diagram reachable from the initial node N. RLVCOMP can recursively be defined as follows:

Figure 4.1:

OPERATION	EFFECT ON THE RESULT	CHANGES TO THE OPERAND
UNION (both operands)	a	a,m(1)
	d	d,m(1)
	m*	m(1)
INTRSCT (both operands)	a	a,m(0)
	d	d,m(0)
	m*	m(1)
DIFRNCE (left operand)	a	a,m(0)
	d	d,m(0)
	m*	m(1)
DIFRNCE (rigth operand)	a	d,m(0)
	d	a,m(0)
	m*	m(1)
PROJECT	a	a,m(2)
	d	d,m(2)
	m*	m(1)
JOIN	a	a,m(1)
	d	d,m(1)
	m*	m(1)
SELECT	a	a,m(2)
	d	d,m(2)
	m*	m(1)
DIVIDE (left operand)	a	a,m(0)
	d	d,m(0)
	m*	m(1)
DIVIDE (right operand)	a	d,m(2)
	d	a,m(2)
	m*	non-appl.
XCPROD (both operands)	a	a
	d	d
	m*	m(1)

(continued)

Figure 4.1:

(continued from previous page)

OPERATION	EFFECT ON THE RESULT	CHANGES TO THE OPERAND
RCOUNT	a	a,m(2)
	d	d,m(2)
	m*	a,d,m(1)
AVERAGE	a	non-appl.
	d	non-appl.
	m*	a,d,m(2)
MAXTUPL	a	a,m(2)
	d	d,m(2)
	m*	m(1)
MINTUPL	a	a,m(2)
	d	d,m(2)
	m*	m(1)
BRDWTOT	a	a,m(2)
	d	d,m(2)
	m*	a,d,m(1)
CHANGES (both operands)	a	m(0)
	d	d,m(0)
	m*	m(1)

Meaning of the symbols:

- a : addition of tuples.
- d : deletion of tuples.
- m* : modification of one of the "indicating" attributes of the result.
- m(0): modification of any of the attributes of the operand.
- m(1): modification of the attributes of the operand that correspond to the "indicating" attributes of the result.
- m(2): modification of the attributes of the operand involved in the operation.

Figure 4.1: Rules for computing the relevance vectors.

```
function RLVCOMP N RLVL;
   RLV(N)<-RLV(N) or RLVL;
   X<-1;
   for each Y in PRE(N) do
   begin
      RLVCOMP(Y,RULES(OPR(N),RLV(N),EOV(N),X));
      X<-X+1;
   end;
end;
```

When an alerter is placed on a target relation, the function RLVCOMP is called using as parameters the node associated to the target, and a relevance vector which has the first or second bit set to one, depending on whether we want to monitor additions or deletions, while each one of the rest of the bits are set if we want the ASA or DSA detection to be done according to the corresponding attributes.

4.3 Partial Evaluation

We will now complete the definition of the data structure associated to each node of the construction diagram as needed to efficiently perform partial evaluation.

1. DIF(n): For base relations, a vector, similar to the relevance vector, indicating the changes that were made to the relation by an update. The presence of a bit whose value is one indicates that an addition, deletion, or modification to the corresponding attribute, affected at least one tuple of the associated relation.

2. POST(n): A list of all the nodes above n in the construction diagram; it is possible, as we mentioned before, that these nodes belong to different alerters since portions of the construction diagram can be shared by more than one. Each element of this list is formed by: (i) PS: the node above; and (ii) RL: the relevance vector of node n as computed with respect to the relevance vector of the node indicated by the related PS. (It should be noted that this relevance is different from RLV(n), even though the latter is the logical or of the RL relevance vectors in POST(n)).

3. CNT(n): The contents of the relation associated to this node. While the target relation is being evaluated, the contents of both, previous and actual states are reflected here.

4. ALT(n): An alerter indicator; if this node is associated to a target relation, it is so indicated by the number that identifies the associated alerter, otherwise it is zero.

5. FLAGn(n): Three binary flags with the following functions: (FLAG1) to prevent alerter evaluation from being performed twice, because more than one relation was affected by an update; (FLAG2) to avoid double evaluation of the new contents of a virtual relation; and (FLAG3) to avoid double evaluation of the old

contents of a virtual relation.

6. MSG(n): For target relations, either a message to be printed if the alerter is triggered, or instead, because of the same reason, a function to be executed. This function can be defined in any terms of our extended algebra and involve any virtual or base relation existing at the moment.

Figure 4.2 shows the definition and construction diagram of the alerters AL1 and AL2. A portion of this construction diagram is shown in Figure 4.3, where the data structure defined above is detailed. It is assumed that the alerters have just been defined.

These alerters are the expression of: (i) AL1: report NAME and PART# for the suppliers where QTY is more than 500, for any part; and (ii) AL2: report the SUPP# of any new supplier. The set of potentially relevant updates for SUPPLIER is formed by those transactions that generate addition of tuples and/or modification to either SUPP# or NAME, since these updates can result in the triggering of one of the alerters defined above; this set is reflected by the relevance vector RLV(SUPPLIER), which is: (1 0 1 1 0 0). On the other hand, only addition of tuples or changes to the SUPP# can be relevant to AL2, while, additionally, modification of the NAME can be relevant to AL1; this is reflected on the RL vectors in POST(SUPPLIER). Finally,

Figure 4.2:

```
define alerter AL1 =
       monitor for additions,
          indicated by SUPP#,PART#
          then report;
       target =
          join((SUPP#),(SUPP#),
             select((QTY > 500),
                SUP-PART),
             SUPPLIER);
end;


define alerter AL2 =
       monitor for additions,
          indicated by SUPP#,
          then report;
       target =
          project((SUPP#),
             SUPPLIER);
end;
```

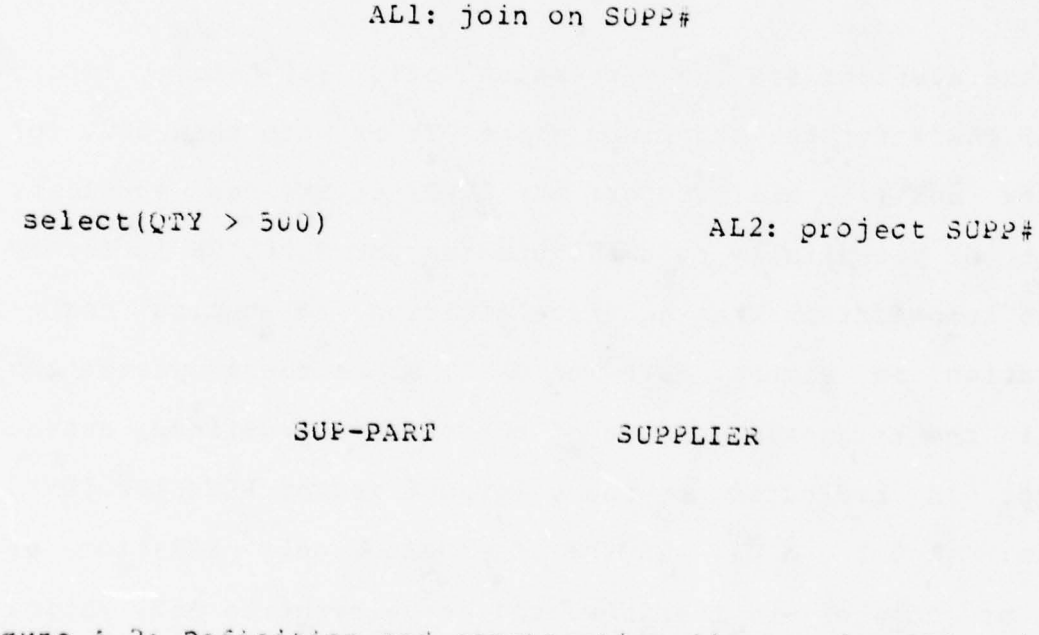

Figure 4.2: Definition and construction diagram for AL1 and AL2

Figure 4.3:

```
                                                        AL2
-------------------------------------------------------------
   POST: [ ^ ] , ALT: #2 , OPR: [ PROJECT [1] ] ,
   RLV: (1 0 1)  ,  DIF: ( ^ )  , CTN: [ ^ ] ,
   FLAG: (0 0 0) , MSG: 'ALERTER #2' , EQV: [1] ,
   PRE: [ SUPPLIER . ]
-------------------------------------------------------------
```

```
                                                    SUPPLIER
-------------------------------------------------------------
   POST: [ [ . (1 0 1 1 0 0)],[ . (1 0 1 0 0 0)] ] ,
   ALT: ^ , OPR: [ BASE ^ ]  , RLV: (1 0 1 1 0 0) ,
   DIF: ( ^ )  ,  CTN: [ SUPPLIER relation ]  ,
   FLAG: (0 0 0)  , MSG: ^ , EQV: ^  , PRE: [ ^ ]
-------------------------------------------------------------
```

Figure 4.3: Construction diagram nodes for SUPPLIER and AL2

OPR(AL2) indicates that the first attribute should be projected from SUPPLIER, while EQV(AL2) shows that the only attribute of AL2 correspond to the first one of its operand.

In order to properly update the relevance vectors in POST(n), the procedure RLVCOMP has to be rewritten as shown below, consequently needing the use of a dummy parameter, (nil), when it is called for a target relation.

```
function RLVCOMP N RLVL ANCS;
   if ANCS.null then
      ALT(N)<-alerter number;
   else for X in POST(N) such that PS(X)=ANCS do
      RL(X)<-RL(X) or RLVL;
   RLV(N)<-RLV(N) or RLVL;
   X<-1;
   for each Y in PRE(N) do;
   begin
      RLVCOMP(Y,RULES(OPR(N),RLV(N),EQV(N),X),ANCS);
      X<-X+1;
   end;
end;
```

We have said that target relations can be evaluated using the construction diagram; in general, the contents of a virtual relation associated to a node in the construction diagram can be evaluated using the next recursive function, which operates on a given node in the construction diagram, and returns the contents of the virtual relation associated to the node.

```
function EVAL N;
   if RATOR(N) is base or FLAGx(N) then
      return CTN(N);
   FLAGx(N)<-1;
   for each X in PRE(N) do
      stack EVAL(X) in REL;
   for each Y in ATTR(N) do
      stack Y in ATR;
   return apply(RATOR(N),(REL,ATR));
end;
```

REL and ATR store temporarily the operands, (relations and their attributes), that will be used by RATOR(N).

In the actual implementation of the function EVAL, the second attribute of the relational operation CHANGES is considered to be making reference to the old contents of a relation, i.e., the state of the relation previous to the update. This way we can use this operation to express a class of alerter predicates based consecutive states of the information in the tuples. As an example: "report any change of salary where the new salary is less than the old salary".

Before going any further, we would like to introduce the notion of transaction. For the purpose of this thesis, we will consider as a transaction to a logically related set of data base updates, which has the characteristic of deferring the evaluation of the alerter until its complete application. For example, in an employee-department data base we could have defined an alerter to report all departments without managers, and another alerter to report all employees who do not belong to an existing department. If we are in a situation when a department is being created, and information for new employees is being loaded, including that of the department manager, it is obvious that any attempt to insert the manager tuple without department information, or vice versa, will lead to the triggering of one of the alerters. We could execute the creation of a new department as a transaction, including the

addition of the department tuple and the addition of the manager tuple, after which the alerter evaluation could be performed. From now on we will consider that the data base is being updated by transactions, and will use the terms updates and transactions indistinctly to refer to data base modifications.

Data base updates can take the form of additions or deletions of tuples, or modification of their attributes. As a consequence of the application of a transaction to the data base, the difference vector, DIF, of the nodes corresponding to the modified base relations are updated to reflect the change that was made. For a modified relation, if a tuple was added and/or deleted, bits one and/or two are respectively set to one, and if for some tuple the kth attribute was modified, then the bit k+2 is set to one.

RIU's can be now identified by direct comparison of the relevance and difference vectors for each base relation; If the logical and of these two vectors does not contain a one, then the transaction is a RIU.

In order to allow the processing of the base relations as a group, the nodes associated to them will be joined in a list. The following function, TRANS-EVAL, will operate on this list, and will start the evaluation of the alerter if the update is not a RIU.

```
function TRANS-EVAL BASELIST;
   for each X in BASELIST do
      if (RLV(X) and DIF(X)) contains a one then
         ALERT-EVAL(X);
end;
```

Once it is determined that a transaction is not a RIU, the function ALERT-EVAL will evaluate the effect of the update and eventually, if it is found relevant will trigger the appropriate alerter. The function ALERT-EVAL can be defined as follows:

```
function ALERT-EVAL N;
   CTN(N)<- BOTH-EVAL(N);
   if not(FLAG1(N)) and CHANGED(CTN(N),RLV(N)) then
      if target(N) then
         trigger(N);
      else
         for each X in POST(N) do
            if CHANGED(PS(X),RL(X)) then
               ALERT-EVAL(PS(X));
   FLAG1(N)<-1;
end;
```

The function BOTH-EVAL evaluates the contents of the previous and actual states of the relation associated to N, using functions equivalent to the function EVAL described before. The function CHANGED takes this result and based on a related relevance vector determines if the changes are relevant, according to the indicated attributes and the first two bits of the vector, (addition and deletion relevance). When the changes are found to be relevant and the node is associated to a target relation, the related alerter is triggered; if a function is specified in MSG(N) then it is executed, otherwise MSG(N) is printed as the standard action, together with the additions or deletions that triggered the alerter. If the changes are

relevant, but the node is not associated to a target, then the relevance of the changes is evaluated for each node in POST(N), and ALERT-EVAL would spread up the construction diagram, evaluating only those nodes relevantly affected by the update.

CHAPTER 5

SUMMARY AND CONCLUSION

In this thesis we have described the algorithms that were used in an implementation of multiple alerters in a relational environment, which efficiently monitor the data base for the presence of certain states that satisfy members of a set of alerter predicates.

For a given data base transaction, the alerters first try to determine, without referencing the contents of the data base, if it is a readily ignorable update, RIU. This is determined by comparing the difference vector generated by the transaction with the relevance vectors of the affected base relations. These relevance vectors define the set of potentially relevant updates. A relevance vector is assigned to each relation, base or virtual, in the construction diagram of the alerters; they are used to control the construction and evaluation of the target relations associated to the alerters.

If an update is not a RIU, then the relations associated to the alerters that can possibly be affected have to be constructed, in order to evaluate how they were modified and, eventually, how does it affect the alerter.

Using the relevance vectors defined for each relation, it is sometimes possible to detect that a transaction will not trigger an alerter even before the target has been completely constructed, since after its construction diagram has been partially evaluated we may determine that the modifications suffered by the virtual relations evaluated to this point are irrelevant to the existing alerters. While partial evaluation is being performed, the relevance vectors are used to detect the addition or deletion of tuples according to the alerters definition. The rules for the computation of these relevance vectors were presented, together with the general design of the algorithms that apply them.

An extended relational algebra was used to express the alerter definitions. In addition to the relational algebra operations defined in [CODD 70], some arithmetical and aggregate capabilities were implemented in order to obtain a much more powerful means of expression for our alerter predicates, and, more important, to study the way in which they can be treated in the context of alerters in relational environments.

Simple and Complex alerters are definable using terms of this extended algebra.

Using the following relations:

EMP : (emp#,dept#,salary) and

DEPT: (dept#,man#)

some examples of complex alerters and of the expressions of their predicates are given below.

1. Structural alerters: "Report any employee who earns more than his manager".

```
target=join((emp#),(man#),
          EMP,
          join((dept#),(dept#),
             EMP,
             DEPT));
```

2. Statistical alerters: "Report when the average salary of a department is over $15000".

```
target=select((average > 15000),
          bkdwtot((salary, by dept#),
             EMP));
```

3. Transition alerters: "Report when the salary of an employee is increased by more than 20%".

```
target=select((salary.new > salary.old+20%),
          changes((emp#),(emp#),
             EMP.new,
             EMP.old));
```

Hammer and Sarin, working in the closely related field of monitoring integrity assertions [HAMM 77], and based on the premise that data base updates can generally be anticipated, introduce an algorithm, the "assertion processor", which

generates procedures for the monitoring of data base integrity assertions. This algorithm, for a given assertion and operation, performs an analysis to determine how the expressions in the assertion can be affected. Based on the result of the analysis, a set of "candidate tests" is produced, which may be used to monitor for assertion violations after a transaction cost estimator has selected the least expensive test.

We feel that there are several aspects which, after some research effort, could further improve the efficiency of the techniques we described here. Construction diagrams are defined by the user in terms of his view of the expression of the alerter predicate. This does not mean that a target relation can only be expressed in one way, since, according to certain rules which can easily be derived, some operations in an algebraic expression could be permuted with no effect on the result. Based on the actual contents of the data base, the operation involved, and the frequency of the same type of transactions, the construction of a target relation could be sensibly more efficient if the target was expressed in an specific way. As an example consider a target relation reflecting the join of a BIRTHPLACE relation with a PERSON relation, restricting the result to those persons over ninety years old, born in New York; this could be either expressed as:

```
target=select((age > 90),
          select((NYers),
             join((on ID),
                BIRTHPLACE,
                PERSON)));
```

or as

```
target=select((NYers),
          join((on ID),
             BIRTHPLACE,
             select((age > 90),
                PERSON)));
```

It is clear that the efficiency of one of those implementations is significantly greater. An appropriate set of rules, involving all or some of the factors mentioned above, could be derived to guide and improve the definition of the construction diagram.

Another aspect that could improve the efficiency of the alerter evaluation is related to the representation of the data base changes, i.e., an extension of the definition of the difference vector, and consequently an extension of the definition of the relevance vector. We have defined these vectors as reflecting changes to their associated attributes, but have excluded any description of the changes. In many cases, mostly involving restriction and arithmetical operations, additional information about the nature of the changes can be useful in the evaluation of the alerter predicate. For example, in the alerter "report any checking account whose balance falls under $100", we should ignore any deposit made to the account, since even though the balance is modified, it can only increase, and therefore the transaction should be recognized as irrelevant to this alerter, and possibly, even treated as RIU's.

A set of rules for the construction and use of these vectors could be formalized in a similar manner to that of Chapter 4, and should considerably reduce the number of occasions in which partial evaluation had to be performed.

The purpose of this Thesis was not to design programs which efficiently handled relational data bases, but rather to explore different alerter techniques. In our case, the relations handled by the programs that were implemented were processed in core, hardly the expected case of an actual implementation. Partial evaluation is the most sensible aspect with respect to this fact, since the efficiency of construction of the virtual relations is directly related to the actual implementation of the relational operators. As we have said before, RIU's can be detected without reference to the data base contents; on the other hand, construction diagrams can be defined and maintained with an expectedly good degree of independence from the actual stored data base, therefore, the efficiency of their operations, partial evaluation excluded, should not vary too much from an implementation to another. It can obviously be concluded here that in the same measure that we improve our capacity of detection of RIU's, (no reference to the data base contents), as much, at least, alerters would improve its efficiency regardless of the environment of implementation.

We feel that the techniques we have described should allow efficient monitoring of alerter predicates in varied relational implementations, and while they have been expressed in relational algebra terms, their use in different language frameworks is not expected to pose a very difficult problem, as the adaptation of the same ideas to the new environment should not be complicated.

APPENDIX A

AN IMPLEMENTATION EXAMPLE

As an illustration of the alerter techniques we have described in this Thesis, we shall now show the results of a test implementation.

Our goal was to monitor the use of the DECSYSTEM-10 computer at the Wharton School. To this effect the files SYSTAT and RELATD were used to represent the following relations:

SYSTAT: (JOB#,NAME,LINE,PROGRAM,ACC#1,ACC#2)

RELATD: (OWNER,NAME)

SYSTAT was being dynamically updated to reflect the use of the DEC-10 computer. For the active jobs, it contained the following information: the job number; user name; the line to which it was attached; the program that was being run; and the account number of the user, represented by two account sub-numbers.

The file RELATD contained a set of owner-username pairs, which defined a relationship between some users and a set of "owners". (let us imagine each owner as a supervisor of the activities of his related users.)

Based on this relations the following alerters were defined:

```
define node ND1=
       project((OWNER,NAME,ACC#1,ACC#2),
          join((NAME),(NAME),
             SYSTAT,
             RELATD));
end;


define alerter AL1=
       monitor for additions,
          indicated by JOB#,ACC#1,ACC#2,
          then report('starting APL program');
       target=
          select((PROGRAM=APL),
             SYSTAT);
end;


define alerter AL2=
       monitor for additions,
          indicated by ACC#1,ACC#2,
          then report('acct with more than one job');
       target=
          select((COUNT > 1),
             rcount((ACC#1,ACC#2),
                SYSTAT));
end;


define alerter AL3=
       monitor for additions,
          indicated by OWNER,ACC#1,ACC#2,
          then report('related user started APL');
       target=
          select((PROGRAM=APL),
             ND1);
end;
```

```
define alerter AL4=
       monitor for additions,
          indicated by all attributes,
          then report('related user logged in');
       target=
          project((OWNER,NAME,ACC#1,ACC#2),
             ND1);
end;
```

For identification purposes, a message was associated to the "report" action of each alerter.

The SYSTAT file was updated by periodically copying it from the general files of the system, it was done based on an specifiable time basis, expressed in terms of seconds. Accordingly, the predicates associated to the alerters were evaluated with a similar periodicity.

We shall now show the printout resulting from the application of these alerters to a normal work day at WCC, when the programs were active for approximately fifteen minutes.

The programs will initially produce a printed image of the alerter definitions that were created. Then they will report the tuples satisfying the alerter predicates when they were initially set; and finally normal triggering of the alerter, due to data changes, is shown.

(The POP-10 code of the programs used in this test implementation will be shown in Appendix B.)

[GHOST TRANSACTION LOG]

```
LOGIN 3056/2
JOB 50 Wharton School KL 603v20 TTY165
[LGNJSP Other jobs same PPN:25,37]
1447        18-Oct-78        Wed

You have old NETMAIL at 20:45 on 17-Oct-78


.r pop

POP10
setpop
: dcomp alerter;


- ALERTER#: 4 - RELEVANCE:[ 1 0 1 1 1 1]
- (OWNER, NAME, ACC#1, ACC#2) - ACCOUNT OF RELATED USER LOGGED IN
 1    <FUNCTION>PROJECT - [[ 1 2 6 7]]
 2       <FUNCTION>JOIN - [[ 2] [ 2]]
 3          <BR> - RELATED
 3          <BR> - SYSTAT

- ALERTER#: 3 - RELEVANCE:[ 1 0 1 0 0 0 0 1 1]
- (OWNER, ACC#1, ACC#2) - RELATED USER STARTS APL PROGRAM
 1    <FUNCTION>SELECT - [[ 5] <FUNCTION>NIL]
 2       <FUNCTION>JOIN - [[ 2] [ 2]]
 3          <BR> - RELATED
 3          <BR> - SYSTAT

- ALERTER#: 2 - RELEVANCE:[ 1 0 1 1 0]
- (ACC#1, ACC#2) - ACCOUNTS WITH MORE THAN ONE JOB
 1    <FUNCTION>SELECT - [[ 1] <FUNCTION>NIL]
 2       <FUNCTION>RCOUNT - [[ 5 6]]
 3          <BR> - SYSTAT

- ALERTER#: 1 - RELEVANCE:[ 1 0 1 0 0 0 1 1]
- (JOB#, ACC#1, ACC#2) - STARTING AN APL PROGRAM
 1    <FUNCTION>SELECT - [[ 4] <FUNCTION>NIL]
 2       <BR> - SYSTAT

<BR> - INDICATES A BASE RELATION


ALERTER # 1 TRIGGERED:
(JOB#, ACC#1, ACC#2) - STARTING AN APL PROGRAM
  4  3070  1013
  10  3000  1337
  12  3000  1341
```

```
  16  3000  1211
  17  3030  141
  18  3001  1002
  19  3000  1117
  20  3000  1313
  23  3003  2005
  24  4325  1
  27  3075  1326
  28  3000  1170
  29  3360  1004
  31  3117  1100
  33  3060  3
  35  3117  1105
  38  3213  1005
  39  3213  1011
  40  3000  1156
  42  3075  1326
  46  3000  1123
  47  3000  1114
  48  4000  43

ALERTER # 2 TRIGGERED:
(ACC#1, ACC#2) - ACCOUNTS WITH MORE THAN ONE JOB
  2  3000
  2  3075
  2  4164
  3  3056
  11  1

ALERTER # 4 TRIGGERED:
(OWNER, NAME, ACC#1, ACC#2) - ACCOUNT OF RELATED USER LOGGED IN
 BUNEMA SERRA  3056  2
 BUNEMA SYSTEM  1  2
 MORGAN SYSTEM  1  2

ALERTER # 2 TRIGGERED:
(ACC#1, ACC#2) - ACCOUNTS WITH MORE THAN ONE JOB
  12  1

ALERTER # 1 TRIGGERED:
(JOB#, ACC#1, ACC#2) - STARTING AN APL PROGRAM
  26  2627  1002

ALERTER # 1 TRIGGERED:
(JOB#, ACC#1, ACC#2) - STARTING AN APL PROGRAM
  48  4000  43

ALERTER # 1 TRIGGERED:
(JOB#, ACC#1, ACC#2) - STARTING AN APL PROGRAM
  21  3000  1123

ALERTER # 4 TRIGGERED:
(OWNER, NAME, ACC#1, ACC#2) - ACCOUNT OF RELATED USER LOGGED IN
```

```
 BUNEMA WEBEL  3000  1036

ALERTER # 2 TRIGGERED:
(ACC#1, ACC#2) - ACCOUNTS WITH MORE THAN ONE JOB
  13  1

ALERTER # 1 TRIGGERED:
(JOB#, ACC#1, ACC#2) - STARTING AN APL PROGRAM
  17  3000  1036
  15  3000  1017

ALERTER # 2 TRIGGERED:
(ACC#1, ACC#2) - ACCOUNTS WITH MORE THAN ONE JOB
  12  1

ALERTER # 3 TRIGGERED:
(OWNER, ACC#1, ACC#2) - RELATED USER STARTS APL PROGRAM
 BUNEMA  3000  1036

ALERTER # 1 TRIGGERED:
(JOB#, ACC#1, ACC#2) - STARTING AN APL PROGRAM
  53  3000  1427
  39  3213  1011
```

APPENDIX B

CONTROL PROGRAMS

- POP-10 CODE -

We shall now show the actual code of the control programs that were used to implement the previous example. The programs are all written as POP-10 functions, and are grouped together, with a not too tight criteria, according to their use.

The function TRANSEVAL, described before, is the starting point for the evaluation of the alerters. After the SYSTAT and RELATD files are modified, this function is referenced using a list that contains the construction diagram nodes associated to the base relations. Each node contains an image of both previous and actual states, from which the modifications suffered by the relations can be evaluated. That would be sufficient, as all other functions would be referenced, as necessary, from this starting point.

```
COMMENT ALERTERS:
        ALERTER EVALUATION FUNCTIONS - SERRA 9/78
-------------------------------------------------;


FUNCTION TRANSEVAL BSELST;
   VARS FLG X1;
      [TIME].POPMESS->FLG;
      BSELST->X1;
      LOOPIF NOT(BSELST.NULL) THEN
         ALERTEVAL(BSELST.HD,FLG);
         BSELST.TL->BSELST;
      CLOSE;
      LOOPIF NOT(X1.NULL) THEN
         X1.HD.NEWR->X1.HD.OLDR;
         X1.TL->X1;
      CLOSE;
END;


FUNCTION ALERTEVAL NODE FLG;
   VARS X1;
      BOTHEVAL(NODE,FLG)->CTNT(NODE);
      IF NOT(FLAG1(NODE)=FLG) AND CHANGED(NODE.CTNT,NODE.RLVNCE)
      THEN
         IF NOT(ALRTSW(NODE)=0)
         THEN
            TRIGGER(NODE);
         ELSE
            NODE.POST->X1;
            LOOPIF NOT(X1.NULL) THEN
               IF CHANGED(NODE.CTNT,X1.HD.RLV)
               THEN
                  ALERTEVAL(X1.HD.PST,FLG);
               CLOSE;
               X1.TL->X1;
            CLOSE;
         CLOSE;
         FLG->FLAG1(NODE);
      CLOSE;
END;


FUNCTION BOTHEVAL NODE FLG;
      CONS(EVALOLD(NODE,FLG),CONS(EVALNEW(NODE,FLG),NIL));
END;
```

```
FUNCTION EVALNEW NODE FLG;
   VARS X1 X2;
      IF NODE.OPRN.RTOR=BASE OR FLG=FLAG3(NODE)
      THEN
         NEWR(NODE);
      ELSE
         FLG->FLAG3(NODE);
         EVALNEW(NODE.PRE.HD,FLG);
         IF NOT(NULL(NODE.PRE.TL))
         THEN
            IF NODE.OPRN.RTOR=CHANGES
            THEN
               EVALOLD(NODE.PRE.TL.HD,FLG);
            ELSE;
               EVALNEW(NODE.PRE.TL.HD,FLG);
            CLOSE;
         CLOSE;
         NODE.OPRN.RAND->X1;
         LOOPIF NOT(NULL(X1)) THEN
            X1.HD;
            X1.TL->X1;
         CLOSE;
         .(NODE.OPRN.RTOR);
      CLOSE;
END;


FUNCTION EVALOLD NODE FLG;
   VARS X1 X2;
      IF NODE.OPRN.RTOR=BASE OR FLG=FLAG2(NODE)
      THEN
         OLDR(NODE);
      ELSE
         FLG->FLAG2(NODE);
         EVALOLD(NODE.PRE.HD,FLG);
         IF NOT(NULL(NODE.PRE.TL))
         THEN
            EVALOLD(NODE.PRE.TL.HD,FLG);
         CLOSE;
         NODE.OPRN.RAND->X1;
         LOOPIF NOT(NULL(X1)) THEN
            X1.HD;
            X1.TL->X1;
         CLOSE;
         .(NODE.OPRN.RTOR);
      CLOSE;
END;
```

```
FUNCTION CHANGED NCTNT NRLVC;
   VARS X1;
      FALSE->X1;
      IF NRLVC.TL.HD
      THEN
         DIFRNCE(PROJECT(NCTNT.HD,ONLIST(NRLVC.TL.TL,1)),
                 PROJECT(NCTNT.TL.HD,ONLIST(NRLVC.TL.TL,1))
                 ).RL.NULL.NOT OR X1 -> X1;
      CLOSE;
      IF NRLVC.HD
      THEN
         DIFRNCE(PROJECT(NCTNT.TL.HD,ONLIST(NRLVC.TL.TL,1)),
                 PROJECT(NCTNT.HD,ONLIST(NRLVC.TL.TL,1))
                 ).RL.NULL.NOT OR X1 -> X1;
      CLOSE;
      X1;
END;


FUNCTION TRIGGER NODE;
      IF DATAWORD(MSG(NODE))="CSTRIP"
      THEN
         2.NL;
         'ALERTER  @.PRSTRING;NODE.ALRTSW.PR;
         ' TRIGGERED:@.PRSTRING;
         1.NL;
         PRSTRING(MSG(NODE));
         1.NL;
         PRRELA(DIFFER(NODE.CTNT,NODE.RLVNCE));
      ELSE
         (NODE.MSG)(NODE);
      CLOSE;
END;


FUNCTION DIFFER NCTNT NRLVC;
      IF NRLVC.TL.HD
      THEN
         DIFRNCE(PROJECT(NCTNT.HD,ONLIST(NRLVC.TL.TL,1)),
                 PROJECT(NCTNT.TL.HD,ONLIST(NRLVC.TL.TL,1)));
      CLOSE;
      IF NRLVC.HD
      THEN
         DIFRNCE(PROJECT(NCTNT.TL.HD,ONLIST(NRLVC.TL.TL,1)),
                 PROJECT(NCTNT.HD,ONLIST(NRLVC.TL.TL,1)));
      CLOSE;
END;
```

```
COMMENT ALERTERS:
        CONSTRUCTION DIAGRAM HANDLING FUNCTIONS - SERRA 9/78
-------------------------------------------------------------;
   VARS BASE PST RLV;
      'BASE RELATION@->BASE;
      RECORDFNS('CONS-DIAGRAM-NODE@,GENLST0(10));
                ->DIFRN  ->MSG     ->FLAG
                ->FEQVL  ->RLVNCE  ->CTNT
                ->OPRN   ->PRE     ->POST
                ->ALRTSW ->CDDEST  ->CDCONS;


FUNCTION CONSCNDG RATOR PAR ERAND;
   VARS X L;
      (NIL,NIL,NIL,NIL,[%NIL,NIL%],
      NIL,NIL,[%NIL,NIL,NIL%],NIL,NIL).CDCONS->X;
      0->ALRTSW(X); PAR->PRE(X);
      CONS(RATOR,CONS(ERAND,NIL))->OPRN(X);
      RLVUPD(X,POST(PAR.HD),GENLST0(LENGTH(PAR.HD.OLDR.DF.RS)+2))
            ->POST(PAR.HD);
      PAR.HD.OLDR.DF::NIL;
      IF NOT(PAR.TL.NULL)
      THEN
         PAR.TL.HD.OLDR.DF::NIL;
         RLVUPD(X,POST(PAR.TL.HD),GENLST0(LENGTH
               (PAR.TL.HD.OLDR.DF.RS)+2))->POST(PAR.TL.HD);
      CLOSE;
      ERAND->L;
      LOOPIF NOT(L.NULL)THEN
         L.HD; L.TL->L;
      CLOSE;
      .RATOR->OLDR(X); OLDR(X)->NEWR(X);
      GENLST0(LENGTH(OLDR(X).DF.RS)+2)->RLVNCE(X);
      FEQVLNCE(RATOR,PAR,ERAND)->FEQVL(X);
      X;
END;


FUNCTION DEFBASE RELL;
   VARS X;
      (NIL,NIL,NIL,NIL,[%NIL,NIL%],
      NIL,NIL,[%NIL,NIL,NIL%],NIL,NIL).CDCONS->X;
      IF RELL.NULL
      THEN
         NIL;
      ELSE
         0->ALRTSW(X);
         RELL.HD->NEWR(X);
```

```
            CONS(RELL.HD.DF,NIL)->OLDR(X);
            CONS(BASE,(NIL::NIL))->OPRN(X);
            CONS(X,DEFBASE(RELL.TL));
            GENLST0(LENGTH(OLDR(X).DF.RS)+2)->RLVNCE(X);
         CLOSE;
END;


FUNCTION RLVCOMP NODE RLVL ANCS ALNO;
   VARS L1 X;
      IF ANCS=NIL
      THEN
         ALNO->ALRTSW(NODE);
      ELSE
         RLVUPD(ANCS,NODE.POST,RLVL)->NODE.POST;
      CLOSE;
      LOR(RLVL,RLVNCE(NODE))->RLVNCE(NODE);
      PRE(NODE)->L1;1->X;
      LOOPIF NOT(L1.NULL) THEN
         RLVCOMP(L1.HD,RULES(NODE.OPRN,RLVL,ELMNT(X,NODE.FEQVL),
               GENLST0(LENGTH(L1.HD.OLDR.DF.RS)),X),NODE,0);
         L1.TL->L1;1+X->X;
      CLOSE;
END;


FUNCTION FEQVLNCE RATOR PRED RANDS;
   VARS L L1 L2;
      0->L1;0->L2;
      LENGTH(PRED.HD.OLDR.DF.RS)->L1;GENLST0(L1)->L;
      IF NOT(PRED.TL.NULL) THEN
         LENGTH(PRED.TL.HD.OLDR.DF.RS)->L2;CLOSE;
      IF LMEMBER(RATOR,[%UNION,INTRSCT,DIFRNCE%]) THEN
         COMPLMNT(NIL,L1,1)::(COMPLMNT(NIL,L2,1)::NIL);EXIT;
      IF LMEMBER(RATOR,[%SELECT,MAXTUPL,MINTUPL%]) THEN
         COMPLMNT(NIL,L1,1)::NIL;EXIT;
      IF RATOR=PROJECT THEN
         [%RANDS.HD%];EXIT;
      IF LMEMBER(RATOR,[%JOIN,CHANGES%]) THEN
         COMPLMNT(NIL,L1,1)::
         (L<>COMPLMNT(RANDS.TL.HD,L2,1)::NIL);EXIT;
      IF RATOR=XCPROD THEN
         COMPLMNT(NIL,L1,1)::(L<>COMPLMNT(NIL,L2,1)::NIL);EXIT;
      IF RATOR=DIVIDE THEN
         COMPLMNT(RANDS.HD,L1,1)::(NIL::NIL);EXIT;
      IF RATOR=AVERAGE THEN
         NIL;EXIT;
      IF RATOR=BKDWTOT  THEN
```

```
        [%RANDS.HD.TL%];EXIT;
    IF RATOR=RCOUNT THEN
        [%0::RANDS.HD%];EXIT;
END;

FUNCTION RLVUPD ANCS LPOS RLVL;
    IF LPOS.NULL
    THEN
        CONS(ANCS::RLVL,NIL);
    ELSEIF ANCS=LPOS.HD.PST
        THEN
            CONS(CONS(LPOS.HD.PST,LOR(RLVL,LPOS.HD.RLV)),LPOS.TL);
        ELSE
            LPOS.HD::(RLVUPD(ANCS,LPOS.TL,RLVL));
    CLOSE;
END;

FUNCTION RULES OPT RLVN FEQ RLV2 X;
    VARS X1 X2 X3 RULE ORTR ORND;
        IF OPT.TL.NULL
        THEN
            NIL->ORND;
        ELSE
            ELMNT(X,OPT.TL.HD)->ORND;
        CLOSE;
        OPT.HD->ORTR;TABSRCH(ORTR,X)->RULE;
        IF NOT(RULE.TL.HD=0)
        THEN
            IF RULE.TL.HD=1
            THEN
                COMPLMNT(NIL,LENGTH(RLV2),1)->X1;
            ELSE
                ORND->X1;
            CLOSE;
            LOOPIF NOT(X1.NULL) THEN
                1->ELMNT(X1.HD,RLV2);X1.TL->X1;
            CLOSE;
        CLOSE;
        IF NOT(RULE.TL.HD=1)
        THEN
            FEQ->X1;RULE.TL.TL.HD->X2;2->X3;
            LOOPIF NOT(X1.NULL) THEN
                1+X3->X3;
                IF NOT(X1.HD=0)
                THEN
                    IF (X2 AND ELMNT(X3,RLVN)) OR
```

```
                (X2.NOT AND ELMNT(X3,RLVN) AND
                LMEMBER(X1.HD,ORND))
             THEN
                1->ELMNT(X1.HD,RLV2);
             CLOSE;
          CLOSE;
          X1.TL->X1;
       CLOSE;
    CLOSE;
    (RULE.HD)(RLVN.HD::(RLVN.TL.HD::NIL))<>RLV2;
END;


FUNCTION TABSRCH ORTR X;
   VARS TABLE;
      [%
      [%BKDWTOT,[%[%LOR1 ,2,0%]%]                    %],
      [%RCOUNT ,[%[%LOR1 ,2,0%]%]                    %],
      [%INTRSCT,[%[%LAND1,1,0%],[%LAND1,1,0%]%]%],
      [%UNION  ,[%[%LAND1,0,1%],[%LAND1,0,1%]%]%],
      [%DIFRNCE,[%[%LAND1,1,0%],[%LNOT ,1,0%]%]%],
      [%SELECT ,[%[%LAND1,2,1%]%]                    %],
      [%DIVIDE ,[%[%LAND1,1,0%],[%LNOT ,1,0%]%]%],
      [%PROJECT,[%[%LAND1,2,0%]%]                    %],
      [%JOIN   ,[%[%LAND1,2,1%],[%LAND1,2,1%]%]%],
      [%AVERAGE,[%[%LOR1 ,2,0%]%]                    %],
      [%MAXTUPL,[%[%LAND1,2,0%]%]                    %],
      [%MINTUPL,[%[%LAND1,2,0%]%]                    %],
      [%CHANGES,[%[%LAND1,1,0%],[%LOR1 ,1,0%]%]%],
      [%XCPROD ,[%[%LAND1,0,1%],[%LAND1,0,1%]%]%]
      %]->TABLE;
      LOOPIF NOT(TABLE.NULL) THEN
         IF X=3
         THEN
            1.NL;TABLE.HD.PR;
            ELSEIF TABLE.HD.HD=ORTR
            THEN
               ELMNT(X,TABLE.HD.TL.HD);EXIT;
            TABLE.TL->TABLE;
      CLOSE;
      NIL;
END;


FUNCTION OLDR X;
      X.CTNT.HD;
END;
```

```
FUNCTION OOLD Y X;
      Y->X.CTNT.HD;
END;


FUNCTION NEWR X;
      X.CTNT.TL.HD;
END;


FUNCTION UNEW Y X;
      Y->X.CTNT.TL.HD;
END;


FUNCTION RTOR XOP;
      XOP.HD;
END;


FUNCTION URTR Y XOP;
      Y->XOP.HD;
END;


FUNCTION RAND XOP;
      XOP.TL.HD;
END;


FUNCTION URND Y XOP;
      Y->XOP.TL.HD;
END;


FUNCTION FLAG1 X;
      FLAG(X).HD;
END;


FUNCTION UF1 Y X;
      Y->X.FLAG.HD;
END;


FUNCTION FLAG2 X;
      FLAG(X).TL.HD;
END;
```

```
FUNCTION UF2 Y X;
      Y->X.FLAG.TL.HD;
END;


FUNCTION FLAG3 X;
      FLAG(X).TL.TL.HD;
END;


FUNCTION UF3 Y X;
      Y->FLAG(X).TL.TL.HD;
END;


UF1->UPDATER(FLAG1);
UF2->UPDATER(FLAG2);
UF3->UPDATER(FLAG3);
URTR->UPDATER(RTOR);
URND->UPDATER(RAND);
UOLD->UPDATER(OLDR);
UNEW->UPDATER(NEWR);
HD->PST;
TL->RLV;
```

```
COMMENT RELATIONAL ALGEBRA OPERATIONS:
        RELATIONAL OPERATORS - SERRA 9/78
-----------------------------------------;


FUNCTION UNION R1 R2;
      RECSORT(R1,NIL)->R1;
      RECSORT(R2,NIL)->R2;
      CONS(R2.DF,UNION1(R1.RL,R2.RL,R1.DF.FN,R2.DF.FN)<>R2.RL);
END;


FUNCTION UNION1 LR1 LR2 LRHF1 LRHF2;
   VARS LR3;
      NIL->LR3;
      IF LR2.NULL
      THEN
         LR1;
      ELSE
         LOOPIF NOT(LR1.NULL) THEN
            IF NOT(RMEMBERS(LR1.HD,LR2,LRHF1.TL.TL,LRHF2.TL.TL))
            THEN
               CONS(LR1.HD,LR3)->LR3;
            CLOSE;
            LR1.TL->LR1;
         CLOSE;
      LR3;
      CLOSE;
END;


FUNCTION INTRSCT R1 R2;
      RECSORT(R1,NIL)->R1;
      RECSORT(R2,NIL)->R2;
      CONS(R1.DF,INTRSCT1(R1.RL,R2.RL,R1.DF.FN.TL.TL,R2.DF.FN.TL.TL));
END;


FUNCTION INTRSCT1 LR1 LR2 LRHF1 LRHF2;
   VARS LR3;
      NIL->LR3;
      LOOPIF NOT(LR1.NULL) THEN
         IF RMEMBERS(LR1.HD,LR2,LRHF1,LRHF2)
         THEN
            CONS(LR1.HD,LR3)->LR3;
         CLOSE;
         LR1.TL->LR1;
      CLOSE;
```

```
        LR3;
END;


FUNCTION DIFRNCE R1 R2;
        RECSORT(R1,NIL)->R1;
        RECSORT(R2,NIL)->R2;
        CONS(R1.DF,DIFRNCE1(R1.RL,R2.RL,R1.DF.FN.TL.TL,R2.DF.FN.TL.TL));
END;


FUNCTION DIFRNCE1 LR1 LR2 LRHF1 LRHF2;
   VARS LR3;
      NIL->LR3;
      IF LR2.NULL
      THEN
         LR1;
      ELSE
         LOOPIF NOT(LR1.NULL) THEN
            IF NOT(RMEMBERS(LR1.HD,LR2,LRHF1,LRHF2))
            THEN
               CONS(LR1.HD,LR3)->LR3;
            CLOSE;
            LR1.TL->LR1;
         CLOSE;
         LR3;
      CLOSE;
END;


FUNCTION PROJECT R RSSLCT;
   VARS RECDEF;
      RECSORT(R,RSSLCT)->R;
      DEFREC1(R.DF.NM,LSELECT(R.DF.RS,RSSLCT))->RECDEF;
      CONS(RECDEF,PROJECT1(R.RL,LSELECT
            (R.DF.FN.TL.TL,RSSLCT),RECDEF.FN));
END;


FUNCTION PROJECT1 LR LRHF1 LRHF2;
   VARS LRHF3 R1 R2 LR3;
      NIL->LR3;
      LOOPIF NOT(LR.NULL) THEN
         LRHF1->LRHF3;
         LOOPIF NOT(LRHF3.NULL) THEN
            (LRHF3.HD)(LR.HD);
            LRHF3.TL->LRHF3;
         CLOSE;
```

```
            (LRHF2.HD)()->R1;
            LRHF2.TL.TL->LRHF3;
            LOOPIF NOT(LR.NULL) AND EQRCOND(R1,LR.HD,LRHF3,LRHF1) THEN
               LR.TL->LR;
            CLOSE;
            CONS(R1,LR3)->LR3;
         CLOSE;
         LR3;
END;


FUNCTION SELECT R FNSLCT CONDFN;
   VARS LR3;
      CONS(R.DF,SELECT1(R.RL,CONDFN,LSELECT(R.DF.FN.TL.TL,FNSLCT)));
END;


FUNCTION SELECT1 LR CONDFN LRHF;
      NIL->LR3;
      LOOPIF NOT(LR.NULL) THEN
         IF CONDFN(LR.HD,LRHF)
         THEN
            CONS(LR.HD,LR3)->LR3;
         CLOSE;
         LR.TL->LR;
      CLOSE;
      LR3;
END;


FUNCTION XCPROD R1 R2;
   VARS RECDEF;
      DEFREC1(R1.DF.NM<=>R2.DF.NM,R1.DF.RS
            <>R2.DF.RS)->RECDEF;
      CONS(RECDEF,XCPROD1(R1.RL,R2.RL,R1.DF.FN.TL.HD,
            R2.DF.FN.TL.HD,RECDEF.FN.HD));
END;


FUNCTION XCPROD1 LR1 LR2 DEST1 DEST2 CONS3;
   VARS LR3;
      NIL->LR3;
      LOOPIF NOT(LR1.NULL) THEN
         (XCPROD2(LR1.HD,LR2,DEST1,DEST2,CONS3)<>LR3)->LR3;
         LR1.TL->LR1;
      CLOSE;
      LR3;
END;
```

```
FUNCTION XCPROD2 R1 LR2 DEST1 DEST2 CONS3;
   VARS LR3;
      NIL->LR3;
      LOOPIF NOT(LR2.NULL) THEN
         DEST1(R1);
         DEST2(LR2.HD);
         CONS(CONS3(),LR3)->LR3;
         LR2.TL->LR2;
      CLOSE;
      LR3;
END;


FUNCTION JOIN R1 R2 FNSLCT1 FNSLCT2;
   VARS FNSLCT3 RECDEF;
      RECSORT(R1,FNSLCT1)->R1;
      RECSORT(R2,FNSLCT2)->R2;
      COMPLMNT(FNSLCT2,LENGTH(R2.DF.RS),1)->FNSLCT3;
      DEFREC1(R1.DF.NM<=>R2.DF.NM,R1.DF.RS
            <> LSELECT(R2.DF.RS,FNSLCT3))->RECDEF;
      CONS(RECDEF,JOIN1(R1.RL,R2.RL,
            LSELECT(R1.DF.FN.TL.TL,FNSLCT1),
            LSELECT(R2.DF.FN.TL.TL,FNSLCT2),
            R1.DF.FN.TL.HD,
            LSELECT(R2.DF.FN.TL.TL,FNSLCT3),
            RECDEF.FN.HD));
END;


FUNCTION JOIN1 LR1 LR2 JRHF1 JRHF2 DEST1 DEST2 CONS3;
   VARS LR3;
      NIL->LR3;
      LOOPIF NOT(LR1.NULL OR LR2.NULL) THEN
         LOOPIF NOT(LR2.NULL) AND
               GTRECORD(LR1.HD,LR2.HD,JRHF1,JRHF2) THEN
            LR2.TL->LR2;
         CLOSE;
         JOIN2(LR1.HD,LR2,JRHF1,JRHF2,DEST1,DEST2,CONS3)<>LR3->LR3;
         LR1.TL->LR1;
      CLOSE;
      LR3;
END;


FUNCTION JOIN2 R1 LR2 JRHF1 JRHF2 DEST1 DEST2 CONS3;
   VARS L LR3;
      NIL->LR3;
      LOOPIF NOT(LR2.NULL) AND NOT(GTRECORD(LR2.HD,R1,JRHF2,JRHF1))THEN
```

```
        IF EQRCOND(R1,LR2.HD,JRHF1,JRHF2)
        THEN
           DEST1(R1);
           DEST2->L;
           LOOPIF NOT(L.NULL) THEN
              (L.HD)(LR2.HD);
              L.TL->L;
           CLOSE;
           CONS(CONS3(),LR3)->LR3;
        CLOSE;
        LR2.TL->LR2;
     CLOSE;
     LR3;
END;


FUNCTION DIVIDE R1 R2 LSLCT1 LSLCT2;
   VARS LSLCT3;
      COMPLMNT(LSLCT1,LENGTH(R1.DF.RS),1)->LSLCT3;
      RECSORT(R1,LSLCT3)->R1;
      RECSORT(R2,LSLCT2)->R2;
      PROJECT(CONS(R1.DF,DIVIDE1(R1.RL,R2.RL,LSELECT(R1.DF.FN.TL.TL,
            LSLCT1),LSELECT(R2.DF.FN.TL.TL,LSLCT2),LSELECT
            (R1.DF.FN.TL.TL,LSLCT3))),LSLCT3);
END;


FUNCTION DIVIDE1 LR1 LR2 LRHF1 LRHF2 DEST1;
   VARS RX L M;
      IF LR1.NULL
      THEN
         NIL;
      ELSE
         LR1.HD->RX;
         NIL->L; LR2->M;
         LOOPIF NOT(LR1.NULL) AND
               EQRCOND(RX,LR1.HD,DEST1,DEST1) THEN
            CONS(LR1.HD,L)->L;
            LR1.TL->LR1;
         CLOSE;
         ALLSORT(L,LERECORD(% LRHF1,LRHF1 %))->L;
         LOOPIF NOT(M.NULL) THEN
            IF NOT(MEMBERS(M.HD,L,LRHF2,LRHF1))
            THEN
               DIVIDE1(LR1,LR2,LRHF1,LRHF2,DEST1);
            EXIT;
            M.TL->M;
         CLOSE;
```

```
        CONS(RX,DIVIDE1(LR1,LR2,LRHF1,LRHF2,DEST1));
    CLOSE;
END;


FUNCTION AVERAGE R FNSLCT;
   VARS L RECDEF;
      DEFREC1(R.DF.NM<=>'-AVRGE@,[0 0 0])->RECDEF;
      AVERAGE1(R.RL,LSELECT(R.DF.FN.TL.TL,FNSLCT))->L;
      IF L.TL.HD=0
      THEN
         (-1)->L.TL.HD;
      CLOSE;
      CONS(RECDEF,CONS((RECDEF.FN.HD)
            ((L.HD)/(L.TL.HD),L.HD,L.TL.HD),NIL));
END;


FUNCTION AVERAGE1 LR LRHF;
   VARS L;
      CONS(0,CONS(0,NIL))->L;
      LOOPIF NOT(LR.NULL) THEN
         (LRHF.HD)(LR.HD)+L.HD->L.HD;
         1+L.TL.HD->L.TL.HD;
         LR.TL->LR;
      CLOSE;
      L;
END;


FUNCTION MAXTUPL R LSLCT;
   VARS L;
      IF LSLCT.NULL
      THEN
         R.DF.FN.TL.TL->L;
      ELSE
         LSELECT(R.DF.FN.TL.TL,LSLCT)->L;
      CLOSE;
      CONS(R.DF,MAXTUPL1(R.RL,L));
END;


FUNCTION MAXTUPL1 LR LRHF;
   VARS L;
      IF LR.NULL
      THEN
         NIL;EXIT;
      LR.HD::NIL->L;
```

```
        LOOPIF NOT(LR.NULL) THEN
            IF LERECORD(L.HD,LR.HD,LRHF,LRHF)
            THEN
                LR.HD::NIL->L;
            CLOSE;
            LR.TL->LR;
        CLOSE;
        L;
END;


FUNCTION MINTUPL R LSLCT;
    VARS L;
        IF LSLCT.NULL
        THEN
            R.DF.FN.TL.TL->L;
        ELSE
            LSELECT(R.DF.FN.TL.TL,LSLCT)->L;
        CLOSE;
        CONS(R.DF,MINTUPL1(R.RL,L));
END;


FUNCTION MINTUPL1 LR LRHF;
    VARS L;
        IF LR.NULL
        THEN
            NIL;EXIT;
        LR.HD::NIL->L;
        LOOPIF NOT(LR.NULL) THEN
            IF LERECORD(LR.HD,L.HD,LRHF,LRHF)
            THEN
                LR.HD::NIL->L;
            CLOSE;
            LR.TL->LR;
        CLOSE;
        L;
END;


FUNCTION BKDWTOT R RSSLCT;
    VARS RECDEF;
        RECSORT(R,RSSLCT.TL)->R;
        DEFREC1(R.DF.NM<=>'-TOT@,LSELECT(R.DF.RS,RSSLCT.TL)<>
                [0 0])->RECDEF;
        CONS(RECDEF,BKDWTOT1(R.RL,LSELECT(R.DF.FN.TL.TL,RSSLCT.TL),
                LSELECT(R.DF.FN.TL.TL,RSSLCT.HD::NIL),RECDEF.FN));
END;
```

```
FUNCTION BKDWTOT1 RL LRHF1 LRHF2 LRHF3;
   VARS X0 X1 X2 LRHF4 CNT TOT;
      RL->X0;NIL->X2;
      LOOPIF NOT(X0.NULL) THEN
         1->CNT;(LRHF2.HD)(X0.HD)->TOT;
         X0.TL->X1;
         LOOPIF NOT(X1.NULL) AND EQRCND(X0.HD,X1.HD,LRHF1,LRHF1)THEN
            1+CNT->CNT;
            (LRHF2.HD)(X1.HD)+TOT->TOT;
            X1.TL->X1;
         CLOSE;
         LRHF1->LRHF4;
         LOOPIF NOT(LRHF4.NULL) THEN
            (LRHF4.HD)(X0.HD);
            LRHF4.TL->LRHF4;
         CLOSE;
         CNT;TOT;
         CONS((LRHF3.HD)(),X2)->X2;
         X1->X0;
      CLOSE;
      X2;
END;


FUNCTION RCOUNT R RSSLCT;
   VARS RECDEF;
      IF RSSLCT.HD=0
      THEN
         NIL->RSSLCT;
      CLOSE;
      RECSORT(R,RSSLCT)->R;
      DEFREC1(R.DF.NM<=>'CNT@,0::LSELECT(R.DF.RS,RSSLCT))->RECDEF;
      CONS(RECDEF,RCOUNT1(R.RL,LSELECT(R.DF.FN.TL.TL,RSSLCT),
            RECDEF.FN));
END;


FUNCTION RCOUNT1 LR LRHF1 LRHF2;
   VARS LRHF3 X0 X1 X2;
      IF LRHF1.NULL
      THEN
         (LRHF2.HD)(LENGTH(LR))::NIL;EXIT;
      LR->X0;NIL->X2;
      LOOPIF NOT(X0.NULL) THEN
         1->CNT;
         X0.TL->X1;
         LOOPIF NOT(X1.NULL) AND EQRCND(X0.HD,X1.HD,LRHF1,LRHF1)THEN
            1+CNT->CNT;
```

```
            X1.TL->X1;
         CLOSE;
         LRHF1->LRHF3;
         CNT;
         LOOPIF NOT(LRHF3.NULL) THEN
            (LRHF3.HD)(X0.HD);
            LRHF3.TL->LRHF3;
         CLOSE;
         CONS((LRHF2.HD)(),X2)->X2;
         X1->X0;
      CLOSE;
      X2;
END;


FUNCTION CHANGES R1 R2 LSLCT1 LSLCT2;
      JOIN(
            DIFRNCE(R1,R2),
            DIFRNCE(R2,R1),
            LSLCT1,LSLCT2);
END;
```

```
COMMENT RELATIONAL ALGEBRA OPERATIONS:
        DISK AND CORE RELATION HANDLING FUNCTIONS - SERRA 9/78
-----------------------------------------------------------------;


FUNCTION DEFREC1 RNAME LRS;
   VARS LRSSIZE LRHF X I;
      NIL->LRHF;
      2+LENGTH(LRS)->LRSSIZE;
      RECORDFNS(RNAME,LRS);
      FORALL I 1 1 LRSSIZE;
         ->X;
         CONS(X,LRHF)->LRHF;
      CLOSE;
      CONS(RNAME,CONS(LRS,CONS(LRHF,NIL)));
END;


FUNCTION LOADREL RNAME;
   VARS LRS RECDEF LR LRSSIZE I X CHARRPTR ITEMRPTR RNM ITRP;
      POPMESS("IN"::(RNAME::[.RLT]))->CHARRPTR;
      INCHARITEM(CHARRPTR)->ITRP; ITMRPT(%ITRP%)->ITEMRPTR;
      NIL->LRS;
      ITEMRPTR()->RNM;
      LOOPIF (ITEMRPTR()->X; NOT(EQU(X,'/@'))) THEN
         STICKON(X,LRS)->LRS
      CLOSE;
      DEFREC1(RNM,LRS)->RECDEF;
      LENGTH(LRS)-1->LRSSIZE;
      NIL->LR;
      ITEMRPTR()->X;
      LOOPIF NOT(X=TERMIN) THEN
         X;
         FORALL I 1 1 LRSSIZE;
            ITEMRPTR();
         CLOSE;
         CONS((RECDEF.FN.HD)(),LR)->LR;
         ITEMRPTR()->X;
      CLOSE;
      CONS(RECDEF,LR);
END;


FUNCTION ITMRPT ITRP;
   VARS X;
      ITRP()->X;
      IF DATAWORD(X)="WORD"
      THEN
```

```
            X.WRDCHR;
        ELSE
            X;
        CLOSE;
END;


FUNCTION SAVEREL R FNAME;
    VARS CUCHAROUT OUTF LRS LRL;
        POPMESS("OUT"::(FNAME::[.RLT]))->OUTF;
        OUTF->CUCHAROUT;
        R.DF.NM.PRSTRING;
        R.DF.RS->LRS;
        LOOPIF NOT(LRS.NULL) THEN
            PR(LRS.HD);
            LRS.TL->LRS;
        CLOSE;
        PRSTRING('/@);
        R.RL->LRL;
        LOOPIF NOT(LRL.NULL) THEN
            PRLIST(DATALIST(LRL.HD));
            LRL.TL->LRL;
            1.NL;
        CLOSE;
        POPMESS("CLOSE"::(OUTF::NIL));
END;


FUNCTION GENREC RNAME LRS;
    VARS RECDEF LR LRSSIZE X Z I;
        0->X;
        DEFREC1(RNAME,LRS)->RECDEF;
        LENGTH(LRS)->LRSSIZE;
        NIL->LR;
        'ENTER DATA AS:@.PRSTRING;
        LRS.PR;
        ' FOLLOWED BY EOF SIGNAL@.PRSTRING;
        POPMESS([PROMPT 'DATA: @])->Z;
        1.NL;
        LOOPIF NOT(X) THEN
            FORALL I 1 1 LRSSIZE;
                ITEMREAD();
            CLOSE;
            CONS((RECDEF.FN.HD)(),LR)->LR;
            ITEMREAD()->X;
        CLOSE;
        POPMESS([PROMPT Z])->Z;
        CONS(RECDEF,LR);
```

AD-A067 168 WHARTON SCHOOL PHILADELPHIA PA DEPT OF DECISION SCIENCES F/G 9/2
CONTROL PROGRAMS AND DATA STRUCTURES FOR A MULTIPLE ALERTER IMP--ETC(U)
DEC 78 C J SERRA N00014-75-C-0462
UNCLASSIFIED 79-03-07 NL

2 OF 2
AD A067168

END
DATE FILMED
6-79
DDC

```
END;

FUNCTION DF R;
      R.HD;
END;

FUNCTION UDF DFL R;
      DFL->R.HD;
END;

FUNCTION NM RECDEF;
      RECDEF.HD;
END;

FUNCTION UNM NMX RECDEF;
      NMX->RECDEF.HD;
END;

FUNCTION RS RECDEF;
      RECDEF.TL.HD;
END;

FUNCTION URS RSL RECDEF;
      RSL->RECDEF.TL.HD;
END;

FUNCTION FN RECDEF;
      RECDEF.TL.TL.HD
END;

FUNCTION UFN FNL RECDEF;
      FNL->RECDEF.TL.TL.HD;
END;

FUNCTION RL R;
      R.TL;
END;
```

```
FUNCTION URL RLL R;
      RLL->R.TL;
END;


UDF->UPDATER(DF);
UNM->UPDATER(NM);
URS->UPDATER(RS);
UFN->UPDATER(FN);
URL->UPDATER(RL);


FUNCTION PRRELA R;
   VARS LR;
      R.RL->LR;
      LOOPIF NOT(LR.NULL) THEN
         PRLIST(DATALIST(LR.HD));
         LR.TL->LR;
         1.NL;
      CLOSE;
END;


FUNCTION RECSORT R LSLCT;
   VARS LERECRD L;
      IF LSLCT.NULL
      THEN
         R.DF.FN.TL.TL->L;
      ELSE
         LSELECT(R.DF.FN.TL.TL,LSLCT)->L;
      CLOSE;
      LERECORD(% L,L %)->LERECRD;
      CONS(R.DF,ALLSORT(R.RL,LERECRD));
END;


FUNCTION EQRCCND R1 R2 LRHF1 LRHF2;
      IF LRHF1.NULL
      THEN
         TRUE;
      ELSEIF EQU((LRHF1.HD)(R1),(LRHF2.HD)(R2))
         THEN
            EQRCCND(R1,R2,LRHF1.TL,LRHF2.TL);
         ELSE
            FALSE;
      CLOSE;
END;
```

```
FUNCTION GTRECORD R1 R2 LRHF1 LRHF2;
      LERECORD(R2,R1,LRHF2,LRHF1) AND
      NOT(EQRCCND(R1,R2,LRHF1,LRHF2));
END;


FUNCTION LERECORD R1 R2 LRHF1 LRHF2;
      IF LRHF1.NULL
      THEN
         TRUE;
      ELSEIF (LRHF1.HD)(R1)=(LRHF2.HD)(R2)
         THEN
            LERECORD(R1,R2,LRHF1.TL,LRHF2.TL);
         ELSEIF ISREAL((LRHF1.HD)(R1)) OR ISINTEGER((LRHF1.HD)(R1))
            THEN
               (LRHF1.HD)(R1)<(LRHF2.HD)(R2);
            ELSEIF EQCHAR((LRHF1.HD)(R1),(LRHF2.HD)(R2))
               THEN
                  LERECORD(R1,R2,LRHF1.TL,LRHF2.TL);
               ELSE
                  LECHAR((LRHF1.HD)(R1),(LRHF2.HD)(R2));
      CLOSE;
END;


FUNCTION RMEMBERS R LR LRHF1 LRHF2;
      LOOPIF NOT(LR.NULL OR GTRECORD(LR.HD,R,LRHF2,LRHF1)) THEN
         IF EQRCCND(R,LR.HD,LRHF1,LRHF2)
         THEN
            TRUE;
         EXIT;
         LR.TL->LR;
      CLOSE;
      FALSE;
END;


FUNCTION RMEMBER R LR;
      LOOPIF NOT(LR.NULL) THEN
         IF EQULIST(DATALIST(R),DATALIST(LR.HD))
         THEN
            TRUE;
         EXIT;
         LR.TL->LR;
      CLOSE;
      FALSE;
END;
```

```
FUNCTION GTRECND R LRHF PAR;
      IF (LRHF.HD)(R) > PAR
      THEN
         TRUE;
      ELSE
         FALSE;
      CLOSE;
END;


FUNCTION LTRECND R LRHF PAR;
      IF (LRHF.HD)(R) < PAR
      THEN
         TRUE;
      ELSE
         FALSE;
      CLOSE;
END;


FUNCTION EQRECND R LRHF PAR;
      IF EQU((LRHF.HD)(R),PAR)
      THEN
         TRUE;
      ELSE
         FALSE;
      CLOSE;
END;


FUNCTION EQLRECND R LRHF LPAR;
      LOOPIF NOT(LPAR.NULL) THEN
         IF EQU((LRHF.HD)(R),LPAR.HD)
         THEN
            TRUE;
         EXIT;
         LPAR.TL->LPAR;
      CLOSE;
      FALSE;
END;


FUNCTION TRUECND R LRHF;
      TRUE;
END;
```

```
COMMENT RELATIONAL ALGEBRA OPERATIONS:
        LIST HANDLING FUNCTIONS - SERRA 9/78
-------------------------------------------;


FUNCTION LSELECT L LSLCT;
      IF LSLCT.NULL
      THEN
         NIL;
      ELSE
         CONS(NSELECT(L,LSLCT.HD),LSELECT(L,LSLCT.TL));
      CLOSE;
END;


FUNCTION LMEMBER X L;
      IF L.NULL
      THEN
         FALSE;
      ELSEIF EQU(X,L.HD)
         THEN
            TRUE;
         ELSE
            LMEMBER(X,L.TL);
      CLOSE;
END;


FUNCTION LDIFRNCE L1 L2;
      IF L1.NULL
      THEN
         NIL;
      ELSEIF LMEMBER(L1.HD,L2)
         THEN
            LDIFRNCE(L1.TL,L2);
         ELSE
            CONS(L1.HD,LDIFRNCE(L1.TL,L2));
      CLOSE;
END;


FUNCTION COMPLMNT L NMAX N;
      IF N>NMAX
      THEN
         NIL;
      ELSEIF LMEMBER(N,L)
         THEN
            COMPLMNT(L,NMAX,N+1);
```

```
            ELSE
               CONS(N,COMPLMNT(L,NMAX,N+1));
         CLOSE;
END;


FUNCTION PRLIST L;
   VARS L1;
      L->L1;
      LOOPIF NOT(L1.NULL) THEN
         1.SP;
         IF ISREAL(L1.HD)
         THEN
            PRREAL(L1.HD,0,2);
         ELSEIF ISCOMPND(L1.HD)
            THEN
               PRSTRING(L1.HD);
            ELSE
               PR(L1.HD);
         CLOSE;
         L1.TL->L1;
      CLOSE;
END;


FUNCTION EQULIST L1 L2;
      IF L1.NULL AND L2.NULL
      THEN
         TRUE;
      ELSEIF L1.NULL OR L2.NULL
         THEN
            FALSE;
         ELSEIF EQU(L1.HD,L2.HD) AND EQULIST(L1.TL,L2.TL)
            THEN
               TRUE;
            ELSE
               FALSE;
      CLOSE;
END;


FUNCTION STICKON X L;
      L<>(X::NIL);
END;
```

```
FUNCTION LOR L1 L2;
      IF L1.NULL
      THEN
         NIL;
      ELSE
         CONS(L1.HD OR L2.HD,LOR(L1.TL,L2.TL));
      CLOSE;
END;


FUNCTION LAND L1 L2;
      IF L1.NULL
      THEN
         NIL;
      ELSE
         CONS(L1.HD AND L2.HD,LAND(L1.TL,L2.TL));
      CLOSE;
END;


FUNCTION LNOT L;
      IF L.NULL
      THEN
         NIL;
      ELSE
         CONS(NOT(L.HD),LNOT(L.TL));
      CLOSE;
END;


FUNCTION ELMNT N X;
      IF N=1
      THEN
         X.HD;
      ELSE
         ELMNT(N-1,X.TL);
      CLOSE;
END;


FUNCTION CHNGELMNT A N X;
      IF N=0
      THEN EXIT;
      IF N=1
      THEN
         A->X.HD;
      ELSE
         CHNGELMNT(A,N-1,X.TL);
```

```
      CLOSE;
END;
CHNGELMNT->UPDATER(ELMNT);


FUNCTION GENLST0 N;
      IF N=0
      THEN
         NIL;
      ELSE
         CONS(0,GENLST0(N-1));
      CLOSE;
END;


FUNCTION LAND1 L;
      LAND(L,[1 1]);
END;


FUNCTION LOR1 L;
      LOR(L,[1 1]);
END;


FUNCTION ONLIST L N;
      IF L.NULL
      THEN
         NIL;
      ELSEIF L.HD
         THEN
            CONS(N,ONLIST(L.TL,N+1));
         ELSE
            ONLIST(L.TL,N+1);
      CLOSE;
END;
```

```
COMMENT RELATIONAL ALGEBRA OPERATIONS:
        BASIC GENERAL FUNCTIONS - SERRA 9/78
-------------------------------------------;


FUNCTION APPENDS S1 S2;
   VARS S12;
      INITC(DATALENGTH(S1)+DATALENGTH(S2))->S12;
      APPENDS1(S1,S12,0);
      APPENDS1(S2,S12,DATALENGTH(S1));
      S12;
END;
      APPENDS->NONOP <=>;


FUNCTION APPENDS1 S1 S2 N;
   VARS J SSIZE;
      DATALENGTH(S1)->SSIZE;
      FORALL J 1 1 SSIZE;
         SUBSCRC(J,S1)->SUBSCRC(J+N,S2);
      CLOSE;
END;


FUNCTION NSELECT L N;
      IF N=1
      THEN
         L.HD;
      ELSE
         NSELECT(L.TL,N-1);
      CLOSE;
END;


FUNCTION EQCHAR S1 S2;
   VARS SL I;
      IF DATALENGTH(S1)=DATALENGTH(S2)
      THEN
         DATALENGTH(S1)->SL;
         FORALL I 1 1 SL;
            IF NOT(SUBSCRC(I,S1)=SUBSCRC(I,S2))
            THEN
               FALSE;
            EXIT;
         CLOSE;
         TRUE;
      ELSE
         FALSE;
```

```
      CLOSE;
END;


FUNCTION LECHAR S1 S2;
   VARS S1L S2L S1X S2X N;
      IF S1=S2
      THEN
         TRUE
      EXIT;
      DATALENGTH(S1)->S1L;DATALENGTH(S2)->S2L;
      1->N;
   L0:
      SUBSCRC(N,S1)->S1X;SUBSCRC(N,S2)->S2X;
      IF S1X=S2X
      THEN
         IF N=S1L
         THEN
            TRUE
         EXIT;
         IF N=S2L
         THEN
            FALSE
         EXIT;
         N+1->N;
         GOTO L0;
      CLOSE;
      S1X<S2X;
END;


FUNCTION EQU X Y;
      IF DATAWORD(X)=DATAWORD(Y) AND DATAWORD(X)="CSTRIP"
      THEN
         EQCHAR(X,Y);
      ELSE
         X=Y;
      CLOSE;
END;


FUNCTION WRDCHR W;
   VARS C N I;
      W.DESTWORD;
      ->N;
      INITC(N)->C;
      FORALL I 1 1 N;
         ->SUBSCRC(N+1-I,C);
```

```
        CLOSE;
        C;
END;
```

BIBLIOGRAPHY

1. [ASTR 76] Astraham, M. M. et al "System R: Relational Approach to Database Management". ACM Transactions on Database Systems, Vol. 1, No. 2, June 1976.

2. [BUNE 76] Buneman, O. P. and Clemons, E. K. "Efficiently Monitoring Relational Data Bases". Working paper 76-10-08, Department of Decision Sciences, The Wharton School, University of Pennsylvania, 1976.

3. [BUNE 75] Buneman, O. P. and Morgan, H. L. "Implementing Alerting Techniques in Database Systems". Working paper 77-03-04 Department of Decision Sciences, The Wharton School, University of Pennsylvania, 1975, revised 1977.

4. [BURS 71] Burstall, R. M., Collins, J. S. and Popplestone, R. J. "Programming in POP-2". Edimburgh University Press, 1971.

5. [CODD 70] Codd, E. F. "A Relational Model of Data for Large Shared Data Banks". Communications of the ACM, Vol. 13, No. 6, June 1970.

6. [CODD 72] Codd, E. F. "Relational Completeness of Data Base Sublanguages". Data Base Systems, Courant Computer Science Symposia Series, Vol. 6, Englewood Cliffs, N.J., Prentice Hall 1972.

7. [DATE 75] Date, C. J. "An Introduction to Database Systems". Addison-Wesley Publishing Company, Inc., 1975.

8. [DAVI 74] Davies, D. J. M. "POP-10 Users Manual". Department of Computer Science, University of Western Ontario, London, Ontario, Canada, 1974, revised 1975.

9. [HAMM 76] Hammer, M. and McLeod, D. "A Framework for Data Base Semantic Integrity". Proceedings of the Second International Conference on Software Engineering, San Francisco, California, 1976.

10. [HAMM 77] Hammer, M. and Sarin, S. "Efficient Monitoring of Database Assertions". MIT Laboratory for Computer Science, 1977.

DISTRIBUTION LIST

Department of the Navy - Office of Naval Research

Data Base Management Systems Project

Defense Documentation Center
(12 copies)
Cameron Station
Alexandria, VA 22314

Office of Naval Research
Code 102IP
Arlington, Virginia 22217

Office of Naval Research
Branch Office, Chicago
536 South Clark Street
Chicago, IL 60605

New York Area Office

715 Broadway - 5th Floor
New York, NY 10003

Dr. A. L. Slafkosky
Scientific Advisor
Commandant of the Marine Corps
(Code RD-1)
Washington, DC 20380

Office of Naval Research
Code 458
Arlington, VA 22217

Office of Naval Research
(2 copies)
Information Systems Program
Code 437
Arlington, VA 22217

Office of Naval Research
Branch Office
495 Summer Street
Boston, MA 02210

Office of Naval Research
Branch Office, Pasadena
1030 East Green Street
Pasadena, CA 91106

Naval Research Laboratory
(6 copies)
Technical Information Division
Code 2627
Washington, DC 20375

Office of Naval Research
Code 455
Arlington, VA 22217

Naval Electronics Laboratory Center
Advanced Software Technology Divisi
Code 5200
San Diego, CA 92152

Mr. E. H. Gleissner
Naval Ship Research and
Development Center
Computation & Mathematics Dept.
Bethesda, MD 20084

Captain Grace M. Hopper
NAICOM/MIS Planning Branch
(OP-916D)
Office of Chief of Naval Operations
Washington, DC 20350

Mr. Kim B. Thompson
Technical Director
Information Systems Division
(OP-911G)
Office of Chief of Naval Operations
Washington, DC 20350

Bureau of Library and
Information Science Research
Rutgers - The State University
189 College Avenue
New Brunswick, NJ 08903
Attn: Dr. Henry Voos

Professor Omar Wing
Columbia University
in the City of New York
Dept. of Electrical Engineering
and Computer Science
New York, NY 10027

Defense Mapping Agency
Topographic Center
ATTN: Advanced Technology
Division
Code 41300 (Mr. W. Mullison)
6500 Brookes Lane
Washington, D.C. 20315

Commander, Naval Sea Systems Command
Department of the Navy
Washington, D.C. 20362
ATTENTION: (PMS30611)

Captain Richard L. Martin, USN
Commanding Officer
USS Francis Marion (LPA-249)
FPO New York 09501

Professor Mike Athans
Massachusetts Institute of Technology
Dept. of Electrical Engineering and
Computer Science
77 Mass. Avenue
Cambridge, MA 02139

DISTRIBUTION LIST

Department of the Navy - Office of Naval Research

Data Base Management Systems Project

Defense Documentation
Cameron Station
Alexandria, VA 22314
12 copies

Office of Naval Research
Branch Office, Chicago
536 South Clark Street
Chicago, IL 60605

Office of Naval Research
New York Area Office
715 Broadway - 5th Floor
New York, N.Y. 10003

Dr. A.L. Slafkosky
Scientific Advisor (RD-1)
Commandant of the Marine Corps
Washington D.C. 20380

Office of Naval Research
Code 458
Arlington, VA 22217

Office of Naval Research
Information Systems Program
Code 437
Arlington, VA 22217
2 copies

Office of Naval Research
Branch Office, Boston
495 Summer Street
Boston, MA 02210

Office of Naval Research
Branch Office, Pasadena
1030 East Green Street
Pasadena, CA 91106

Naval Research Laboratory
Technical Information Division
Code 2627
Washington, D.C. 20375
6 copies

Office of Naval Research
Code 455
Arlington, VA 2217

Naval Electronics Lab. Cener
Advanced Software Technology Div.
Code 5200
San Diego, CA 92152

Mr. E. H. Gleissner
Naval Ship Research and
Development Center
Computation and Mathematic Dept.
Bethesda, MD 20084

Mr. Kim Thompson
Technical director
Information Systems Division
OP-911G
Office of Chief Naval Operations
Washington, D.C. 20350

Prof. Omar Wing
Columbia University
in the City of New York
Dept. of Electrical Engineering
and Computer Science
New York, N.Y. 10027

Commander, Naval Sea Systems Command
Department of the Navy
Washington, D.C. 20362
ATTENTION: PMS30611

Captain Richard Martin, USN
Commanding Officer
USS Francis Marion (LPA-249)
FPO New York 09501

Captain Grace M. Hopper
NAICOM/MIS Planning Branch
OP-916D
Office of Chief of Naval Research
Washington, D.C. 20350

Bureau of Library and
Information Science Research
Rutgers - The State University
189 College Avenue
New Brunswick, N.J. 08903
ATTENTION: Dr. Henry Voos

Defense Mapping Agency
Topographic Center
ATTN: Advanced Technology Div.
Code 41300
6500 Brookes Lane
Washington, D.C. 20315